```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
#include <stdio.h>
#include <shastra/network/sharedMem.h>

#ifdef WANT
shmid_ds contains
          struct  ipc_perm shm_perm;  /* operation permission struct */
          int     shm_segsz;          /* size of segment */
          ushort  shm_cpid;           /* creator pid */
          ushort  shm_lpid;           /* pid of last operation */
          short   shm_nattch;         /* number of current attaches */
          time_t  shm_atime;          /* last attach time */
          time_t  shm_dtime;          /* last detach time */
          time_t  shm_ctime;          /* last change time */
                                      /* Times measured in secs since */
                                      /* 00:00:00 GMT, Jan. 1, 1970 */
ipc_perm contains
          ushort  cuid;               /* creator user id */
          ushort  cgid;               /* creator group id */
          ushort  uid;                /* user id */
          ushort  gid;                /* group id */
          ushort  mode;               /* r/w permission */

#endif /*WANT*/

#define ALIGN2FOUR(n) (((n)/4+1)*4)

shmInfo *
shmInfoCreate()
{
    shmInfo *pShmInfo;

    pShmInfo = (shmInfo*)malloc(sizeof(shmInfo));
    memset(pShmInfo, 0, sizeof(shmInfo));
    pShmInfo->shmId = -1;
    pShmInfo->shmAddr = (char*)-1;

    return pShmInfo;
}

int
shMemAlloc(pShmInfo, nSize)
shmInfo *pShmInfo;
int nSize;
{
    /*
    pShmInfo = shmInfoCreate();
    */
    if(!pShmInfo){
        return 0;
    }

    nSize = ALIGN2FOUR(nSize);

    pShmInfo->shmId = shmget(IPC_PRIVATE, nSize, IPC_CREAT|0755);
    if(pShmInfo->shmId < 0) {
        perror("shmget");
        return(0);
    }

    pShmInfo->shmSize = nSize;

    pShmInfo->shmAddr = (char *)shmat(pShmInfo->shmId, 0, 0);
    if(pShmInfo->shmAddr == ((char *)-1)) {
        perror("shmat");
        return(0);
    }

    /* Clear the memory out */
    memset(pShmInfo->shmAddr, 0, nSize);

    return 1;
}



int
shMemConnect(pShmInfo)
shmInfo *pShmInfo;
{
    if(!pShmInfo || (pShmInfo->shmId < 0)){
        return 0;
    }
    pShmInfo->shmAddr = (char *)shmat(pShmInfo->shmId, 0, 0);

    if(pShmInfo->shmAddr == ((char *)-1)) {
        perror("shmat");
        return(0);
    }
    if(shMemGetInfo(pShmInfo) != 0){
        pShmInfo->shmSize = pShmInfo->shmIdDS.shm_segsz;
    }
    return 1;
}


int
shMemDisconnect(pShmInfo)
shmInfo *pShmInfo;
{
    if(!pShmInfo || (pShmInfo->shmId < 0) || (pShmInfo->shmAddr == (char*)-
        1)){
        return 0;
    }
    if(shMemGetInfo(pShmInfo) != 0){
        if(getpid() == pShmInfo->shmIdDS.shm_cpid){
            shMemFree(pShmInfo);
            if(  pShmInfo->shmIdDS.shm_nattch > 1){
                fprintf(stderr,
                    "shMemDisconnect()->warning.. %d procs still attached!\
                        n",
                    pShmInfo->shmIdDS.shm_nattch);
            }
        }
    }
    if(pShmInfo->shmAddr != (char*)-1){
        if(shmdt(pShmInfo->shmAddr) == -1){
            perror("shmdt");
            pShmInfo->shmAddr = (char*)-1;
            return(0);
        }
        pShmInfo->shmAddr = (char*)-1;
    }
    return 1;
}

int
shMemReconnect(pShmInfo, shmId)
shmInfo *pShmInfo;
int shmId;
{
    if(!pShmInfo || (shmId < 0)){
        return 0;
    }
    if(pShmInfo->shmId != shmId){
        shMemDisconnect(pShmInfo);
        pShmInfo->shmId = shmId;
        return shMemConnect(pShmInfo);
    }
    return 1;
}

int
shMemDelete(pShmInfo, shmId)
shmInfo *pShmInfo;
int shmId;
{
    if(!pShmInfo || (shmId < 0)){
        return 0;
    }
    if(pShmInfo->shmId == shmId){
        return shMemFree(pShmInfo);
    }
    return 0;
}

int
shMemFree(pShmInfo)
shmInfo *pShmInfo;
{
    if(!pShmInfo || (pShmInfo->shmId < 0)){
        return 0;
    }
    if(pShmInfo->shmAddr != (char*)-1){
        if(shmdt(pShmInfo->shmAddr) == -1){
            perror("shmdt");
            pShmInfo->shmAddr = (char*)-1;
            return(0);
        }
    }
    if(shmctl(pShmInfo->shmId, IPC_RMID, NULL) == -1){
        perror("shmctl(IPC_RMID)");
        return(0);
    }
    pShmInfo->shmId = -1;
    pShmInfo->shmAddr = (char*)-1;

    return 1;
}



int
shMemGetInfo(pShmInfo)
shmInfo *pShmInfo;
{
    if(!pShmInfo || (pShmInfo->shmId < 0)){
        return 0;
    }
```

```c
    if(shmctl(pShmInfo->shmId, IPC_STAT, &pShmInfo->shmIdDS) == -1){
        perror("shmctl(IPC_STAT)");
        return(0);
    }
    return 1;
}


int
shMemReuseSegment(pShmInfo, nSize)
shmInfo *pShmInfo;
int nSize;
{
    if(!pShmInfo ){
        return 0;
    }
    if(pShmInfo->shmId >= 0){
        if(nSize > pShmInfo->shmSize){
            shMemDisconnect(pShmInfo);
            return shMemAlloc(pShmInfo, nSize);
        }
    }
    else{
        return shMemAlloc(pShmInfo, nSize);
    }
    return 1;
}
```

```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/*
 * test.c -- multicast testing
 */

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <netdb.h>
#include <sys/time.h>
#include <sys/file.h>
#include <sys/types.h>
#ifdef SHASTRA4SUN5
#include <sys/systeminfo.h>
#include <sys/sockio.h>
#endif
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <net/if.h>

#include <shastra/network/mplex.h>
#include <shastra/network/udp.h>

/*UDP utils
 use connect to isolate comm endpoint, and bad connect to disconnect
 or good connect to reconnect elsewhere
 */
```

```
/*
 valid Sun4.1 net interfaces (akhil etc:
  le0, lo0  (ell ee zero, ell oh zero)
 valid SGI net interfaces (arjun, agasti etc:
  ec0, lo0  (ee cee zero, ell oh zero)
  (escher)
  et0, fxp0, lo0 (ee tee zero, eff ex pee zero, ell oh zero)
 */

static int cmGetMulticastInterface(Prot4(char*, char*, int, struct in_addr*
    ));
static int cmGetBroadcastInterface(Prot5(char*, char*, int, struct in_addr*
    ,
    struct sockaddr_in*));
static int cmConvertString2IPAddress(Prot2( char *, struct in_addr *));

static struct sockaddr_in  saInMine;

static int
cmConvertString2IPAddress(sIFAddr, pInAddrIF)
     char *sIFAddr;
     struct in_addr *pInAddrIF;
{
  struct hostent *pheHost;

  if (sIFAddr == NULL){
    return 0;
  }
  pInAddrIF->s_addr = inet_addr(sIFAddr);
  if (pInAddrIF->s_addr == (unsigned long)-1){
    pheHost = gethostbyname(sIFAddr);
    if (pheHost != NULL){
      memcpy(pInAddrIF, pheHost->h_addr, pheHost->h_length);
    }
    else{
      fprintf(stderr, "cmConvertString2IPAddress() No IP address for '%s'\
          n",
          sIFAddr);
      return(-1);
    }
  }
  return 0;
}


/*
 * dump info about network interfaces
 */
static void
cmShowInterfaces(iFd)
  int iFd;
{
  int i;
  struct ifconf   ifConf;
  struct ifreq    *pIFReq ;
  char            sbBuffer[BUFSIZ] ;
  struct sockaddr_in *pSockAddr;

  ifConf.ifc_len = sizeof( sbBuffer ) ;
  ifConf.ifc_buf = sbBuffer ;
  if( ioctl( iFd, SIOCGIFCONF, (char *) &ifConf ) < 0 ) {
    perror( "ioctl() SIOCGIFCONF" ) ;
    return;
  }

  pIFReq = ifConf.ifc_req;
  for( i = ifConf.ifc_len/sizeof(*pIFReq) ; --i >= 0 ; pIFReq++ ) {
    pSockAddr = (struct sockaddr_in*)&pIFReq->ifr_addr;
    fprintf(stderr, "Interface[%d] - %s, Flags(%d, 0x%x), \
Family:%d, Address:%ld (0x%lx)\n",
        i, pIFReq->ifr_name, pIFReq->ifr_flags, pIFReq->ifr_flags,
        pSockAddr->sin_family,
        pSockAddr->sin_addr.s_addr, pSockAddr->sin_addr.s_addr);

  }
}

/*
 * get/check if interface exists and is capable of doing multicasting.
 */
static int
cmGetMulticastInterface(sIFAddr, sInterface, iFd, pInAddrIF)
     char *sIFAddr;
     char *sInterface;
     int iFd;
     struct in_addr *pInAddrIF;
{
#ifdef HAVEMULTICAST
  int             i, fFound;
  struct ifconf   ifConf;
  struct ifreq    *pIFReq ;
  struct in_addr inAddrIF;
  char            sbBuffer[BUFSIZ] ;
  char *sLocal;

  if( sIFAddr != NULL) {
    if( cmConvertString2IPAddress(sIFAddr, &inAddrIF) < 0){
      inAddrIF.s_addr = INADDR_ANY;
    }
  }
  else{
    inAddrIF.s_addr = INADDR_ANY;
  }

  ifConf.ifc_len = sizeof( sbBuffer ) ;
  ifConf.ifc_buf = sbBuffer ;
  if( ioctl( iFd, SIOCGIFCONF, (char *) &ifConf ) < 0 ) {
    perror( "ioctl() SIOCGIFCONF" ) ;
    return( -1 ) ;
  }

  fFound = 0;
  pIFReq = ifConf.ifc_req;
  for( i = ifConf.ifc_len/sizeof(*pIFReq) ; --i >= 0 ; pIFReq++ ) {
    fprintf(stderr, "Interface[%d] - %s, INET=%d, MCAST=%d, flags=%d\n",
            i, pIFReq->ifr_name, pIFReq->ifr_addr.sa_family == AF_INET,
            pIFReq->ifr_flags & IFF_MULTICAST, pIFReq->ifr_flags);
    if( pIFReq->ifr_addr.sa_family != AF_INET ){
      continue ;
    }
    if( !( pIFReq->ifr_flags & IFF_MULTICAST ) ) {
      continue ;
    }

    if(sInterface == NULL){
      sLocal = pIFReq->ifr_name;
    }
    else{
      sLocal = sInterface;
    }
    if( strncmp( pIFReq->ifr_name, sLocal, strlen( pIFReq->ifr_name ) )
        == 0 ) {
      fFound = 1;
      *pInAddrIF = ((struct sockaddr_in *) &pIFReq->ifr_addr)->sin_addr ;

      if( ioctl( iFd, SIOCGIFFLAGS, (char *) pIFReq ) < 0 ) {
        perror( "ioctl() SIOCGIFFLAGS" ) ;
        return( -1 ) ;
      }

      if(pInAddrIF->s_addr == INADDR_ANY ) {
        fprintf(stderr, "cmGetMulticastInterface()->%s: invalid interface
            address\n", sLocal);
        return( -1 ) ;
      }
      if((inAddrIF.s_addr != INADDR_ANY) &&
     (pInAddrIF->s_addr != inAddrIF.s_addr)){
    continue;
      }
      break;
    }
  }
  if( !fFound) {
    if(sInterface != NULL){
      fprintf(stderr, "cmGetMulticastInterface()->%s: unknown interface\n",
          sInterface);
    }
    else{
      fprintf(stderr, "cmGetMulticastInterface()->no interface\n");
    }
    return( -1 ) ;
  }
  return( 0 ) ;
#else               /* HAVEMULTICAST*/
  return -1;
#endif              /* HAVEMULTICAST*/
}

/*
 * get/check if interface exists and is capable of doing broadcasting.
 */
static int
cmGetBroadcastInterface(sIFAddr, sInterface,  iFd, pInAddrIF, pSockAddr)
     char *sIFAddr;
     char *sInterface;
     int iFd;
     struct in_addr *pInAddrIF;
     struct sockaddr_in *pSockAddr;
{
  int               i, fFound ;
  struct in_addr inAddrIF;
  struct ifconf   ifConf;
  struct ifreq    *pIFReq ;
  char              sbBuffer[BUFSIZ] ;
  char *sLocal;

  if( sIFAddr != NULL) {
    if( cmConvertString2IPAddress(sIFAddr, &inAddrIF) < 0){
      inAddrIF.s_addr = INADDR_ANY;
    }
  }
  else{
    inAddrIF.s_addr = INADDR_ANY;
  }
  ifConf.ifc_len = sizeof( sbBuffer ) ;
  ifConf.ifc_buf = sbBuffer ;
  if( ioctl( iFd, SIOCGIFCONF, (char *) &ifConf ) < 0 ) {
    perror( "ioctl() SIOCGIFCONF" ) ;
    return( -1 ) ;
  }

  fFound = 0;
  pIFReq = ifConf.ifc_req;
  for( i = ifConf.ifc_len/sizeof(*pIFReq) ; --i >= 0 ; pIFReq++ ) {
    fprintf(stderr, "Interface[%d] - %s, INET=%d, BCAST=%d, flags=%d\n",
            i, pIFReq->ifr_name, pIFReq->ifr_addr.sa_family == AF_INET,
            pIFReq->ifr_flags & IFF_BROADCAST, pIFReq->ifr_flags);
    if( pIFReq->ifr_addr.sa_family != AF_INET ){
      continue ;
    }
    if(!( pIFReq->ifr_flags & IFF_BROADCAST)){
      continue ;
    }
    if(sInterface == NULL){
      sLocal = pIFReq->ifr_name;
    }
    else{
      sLocal = sInterface;
    }
    if( strncmp( pIFReq->ifr_name, sLocal, strlen( pIFReq->ifr_name ) )
    == 0 ) {
      fFound = 1;
      *pInAddrIF = ((struct sockaddr_in *) &pIFReq->ifr_addr)->sin_addr ;
      if( pInAddrIF->s_addr == INADDR_ANY ) {
        fprintf(stderr, "cmGetBroadcastInterface() ->%s: invalid interface
            address\n", sLocal);
        return( -1 ) ;
      }
      if((inAddrIF.s_addr != INADDR_ANY) &&
     (pInAddrIF->s_addr != inAddrIF.s_addr)){
    continue;
      }
      if( ioctl( iFd, SIOCGIFFLAGS, (char *) pIFReq ) < 0 ) {
        perror( "ioctl() SIOCGIFFLAGS" ) ;
        return( -1 ) ;
      }
      if( ioctl( iFd, SIOCGIFBRDADDR, (char *) pIFReq ) < 0 ) {
        perror( "ioctl() SIOCGIFBRDADDR" ) ;
        return( -1 ) ;
      }
      memcpy(pSockAddr, &pIFReq->ifr_broadaddr, sizeof(pIFReq->
          ifr_broadaddr));
      break;
    }
  }
  if( !fFound ) {
    if(sInterface){
      fprintf(stderr, "cmGetBroadcastInterface()->%s: unknown interface\n",
          sInterface);
    }
    else{
      fprintf(stderr, "cmGetBroadcastInterface()->no interface\n");
    }
    return( -1 ) ;
  }
  return( 0 ) ;
}

/*
 * Get a unicast socket for the given service.
 */

int
cmSetupUCastSocket(sService, iPort, eSockMode, pSockAddr)
     char *sService;
     int iPort;
     enum udpSockMode eSockMode;
     struct sockaddr_in *pSockAddr;
{
  struct hostent *pheHost;
  struct servent *pseService;
  int iFd, iRetVal;
  unsigned char cUtil;
  unsigned short hUtil;
  unsigned int iUtil;

  switch(eSockMode){
  case udpRead:
  case udpWrite:
  case udpReadWrite:
    break;
  default:
    fprintf( stderr, "Invalid udp mode %d\n", eSockMode) ;
    return( -1 ) ;
  }

  memset(pSockAddr, 0, sizeof(*pSockAddr));
  pSockAddr->sin_addr.s_addr = INADDR_ANY;
  pSockAddr->sin_family = AF_INET;

  if(sService != NULL){
    pseService = getservbyname(sService, "udp");
    if (pseService == NULL){
      fprintf(stderr, "Can't find udp service \"%s\"\n", sService);
      return(-1);
    }
    pSockAddr->sin_port = pseService->s_port;
  }
  else{
    hUtil = iPort;
    pSockAddr->sin_port = htons(hUtil);
  }

  iFd = socket(AF_INET, SOCK_DGRAM, 0);
  if (iFd < 0){
    perror("socket()");
    return(-1);
  }

  switch(eSockMode){
  case udpRead:
  case udpReadWrite:
    iUtil = 1;
    if(setsockopt(iFd, SOL_SOCKET, SO_REUSEADDR, &iUtil, sizeof(iUtil))
       < 0 ) {
      perror( "setsockopt() SOL_SOCKET SO_REUSEADDR" ) ;
      close( iFd ) ;
      return( -1 ) ;
    }
#ifdef SO_REUSEPORT
    if(setsockopt(iFd, SOL_SOCKET, SO_REUSEPORT, &iUtil, sizeof(iUtil))
       < 0 ) {
      perror( "setsockopt() SOL_SOCKET SO_REUSEPORT" ) ;
      close( iFd ) ;
      return( -1 ) ;
    }
#endif              /* SO_REUSEPORT */
    if (bind(iFd, pSockAddr, sizeof(*pSockAddr)) < 0){
      perror("bind()");
      close(iFd);
      return(-1);
    }
    if(eSockMode == udpRead){
      break;
    }
    /*fall-thru for udpReadWrite */
  case udpWrite:
    break;
  }

#ifdef WANT_FIONBIO
  cUtil = 1;
  if (ioctl(iFd, FIONBIO, &cUtil) < 0){
    perror("ioctl() FIONBIO");
    close(iFd);
    return(-1);
  }
#else               /*WANT_FIONBIO*/
  if( fcntl( iFd, F_SETFL, FNDELAY ) < 0 ) {
    perror( "fcntl() F_SETFL FNDELAY" ) ;
    close(iFd);
    return( -1 ) ;
  }
#endif              /*WANT_FIONBIO*/
  return(iFd);
}

/*
 * Get a broadcast socket for the given service.
 */

int
cmSetupBCastSocket(sService, iPort, sIFAddr, sInterface, eSockMode,
    pSockAddr)
     char *sService;
     int iPort;
     char *sIFAddr;
     char *sInterface;
     enum udpSockMode eSockMode;
     struct sockaddr_in *pSockAddr;
{
    struct hostent *pheHost;
    struct servent *pseService;
    struct in_addr inAddrIF;
    int iFd, iRetVal;
    unsigned char cUtil;
    unsigned short hUtil;
    unsigned int iUtil;

    switch(eSockMode){
    case udpRead:
    case udpWrite:
    case udpReadWrite:
      break;
    default:
      fprintf( stderr, "Invalid udp mode %d\n", eSockMode) ;
      return( -1 ) ;
    }

    memset(pSockAddr, 0, sizeof(*pSockAddr));
    pSockAddr->sin_addr.s_addr = INADDR_ANY;
    pSockAddr->sin_family = AF_INET;

    if(sService != NULL){
      pseService = getservbyname(sService, "udp");
      if (pseService == NULL){
        fprintf(stderr, "Can't find udp service \"%s\"\n", sService);
        return(-1);
      }
      pSockAddr->sin_port = pseService->s_port;
    }
    else{
      hUtil = iPort;
      pSockAddr->sin_port = htons(hUtil);
    }

    iFd = socket(AF_INET, SOCK_DGRAM, 0);
    if (iFd < 0){
      perror("socket()");
      return(-1);
    }

    switch(eSockMode){
    case udpRead:
    case udpReadWrite:
      iUtil = 1;
      if(setsockopt(iFd, SOL_SOCKET, SO_REUSEADDR, &iUtil, sizeof(iUtil))
         < 0 ) {
        close( iFd ) ;
        perror( "setsockopt() SOL_SOCKET SO_REUSEADDR" ) ;
        return( -1 ) ;
      }
  #ifdef SO_REUSEPORT
      if(setsockopt(iFd, SOL_SOCKET, SO_REUSEPORT, &iUtil, sizeof(iUtil))
       < 0 ) {
      close( iFd ) ;
      perror( "setsockopt() SOL_SOCKET SO_REUSEPORT" ) ;
      return( -1 ) ;
    }
#endif              /* SO_REUSEPORT */
    if (bind(iFd, pSockAddr, sizeof(*pSockAddr)) < 0){
      perror("bind()");
      close(iFd);
      return(-1);
    }
    if(eSockMode == udpRead){
      break;
    }
    /*fall-thru for udpReadWrite */
  case udpWrite:
    /* TESTING -- pSockAddr->sin_addr.s_addr = INADDR_LOOPBACK; return;*/
/*new broadcast method, not yet on our sun4.1*/
    if(
#if defined SHASTRA4SGI || defined SHASTRA4SUN5 || defined SHASTRA4HP
       sInterface || sIFAddr
#else               /*SHASTRA4SUN4*/
       TRUE
#endif              /*SHASTRA4SUN4*/
       ) {
      iRetVal = cmGetBroadcastInterface( sIFAddr, sInterface, iFd, &
          inAddrIF,
                       pSockAddr);
      if(iRetVal < 0){
        close( iFd ) ;
        return( iRetVal ) ;
      }
      if(sService != NULL){
        pSockAddr->sin_port = pseService->s_port;
      }
      else{
        hUtil = iPort;
        pSockAddr->sin_port = htons(hUtil);
      }
    }
    else{
      pSockAddr->sin_addr.s_addr = INADDR_BROADCAST;
    }
    iUtil = 1;
    if (setsockopt(iFd, SOL_SOCKET, SO_BROADCAST, &iUtil,
           sizeof (iUtil)) < 0){
      perror("setsockopt() SOL_SOCKET SO_BROADCAST");
      close(iFd);
      return(-1);
    }
    break;
  }
```

```
#ifdef WANT_FIONBIO
  cUtil = 1;
  if (ioctl(iFd, FIONBIO, &cUtil) < 0){
    perror("ioctl() FIONBIO");
    close(iFd);
    return(-1);
  }
#else               /*WANT_FIONBIO*/
  if( fcntl( iFd, F_SETFL, FNDELAY ) < 0 ) {
    perror( "fcntl() F_SETFL FNDELAY" ) ;
    close(iFd);
    return( -1 ) ;
  }
#endif              /*WANT_FIONBIO*/
  return(iFd);
}

/*
 * Get a multicast socket for the given service.
 */

int
cmSetupMCastSocket(sService, iPort, sIFAddr, sInterface, sGrpAddr,
         iTTL, fLoopBack, eSockMode, pSockAddr)
     char *sService;
     int iPort;
     char *sIFAddr;
     char *sInterface;
     char *sGrpAddr;
     int iTTL;
     int fLoopBack;
     enum udpSockMode eSockMode;
     struct sockaddr_in *pSockAddr;
{
#ifdef HAVEMULTICAST
  struct ip_mreq ipMRequest;
  struct in_addr inAddrGrp;
  struct in_addr inAddrIF;
  struct hostent *pheHost;
  struct servent *pseService;
  int iFd, iRetVal, iLen;
  unsigned char cUtil;
  unsigned short hUtil;
  unsigned int iUtil;

  memset(&inAddrGrp, 0, sizeof(inAddrGrp));
  inAddrGrp.s_addr = inet_addr( sGrpAddr ) ;
  if( !IN_MULTICAST( inAddrGrp.s_addr ) ) {
    fprintf( stderr, "Invalid multicast address: %s\n", sGrpAddr ) ;
    return( -1 ) ;
  }

  switch(eSockMode){
    case udpRead:
    case udpWrite:
    case udpReadWrite:
      break;
    default:
      fprintf( stderr, "Invalid udp mode %d\n", eSockMode) ;
      return( -1 ) ;
    }

    memset(pSockAddr, 0, sizeof(*pSockAddr));
    pSockAddr->sin_addr.s_addr = INADDR_ANY;
    pSockAddr->sin_family = AF_INET;

    if(sService != NULL){
      pseService = getservbyname(sService, "udp");
      if (pseService == NULL){
        fprintf(stderr, "Can't find udp service \"%s\"\n", sService);
        return(-1);
      }
      pSockAddr->sin_port = pseService->s_port;
    }
    else{
      hUtil = iPort;
      pSockAddr->sin_port = htons(hUtil);
    }

    iFd = socket(AF_INET, SOCK_DGRAM, 0);
    if (iFd < 0){
      perror("socket()");
      return(-1);
    }

    memset(&inAddrIF, 0, sizeof(inAddrIF));
    inAddrIF.s_addr = INADDR_ANY;
  /*new mcast not yet on suns*/
    if(sIFAddr || sInterface) {
      iRetVal = cmGetMulticastInterface( sIFAddr, sInterface, iFd, &inAddrIF)
          ;
      if(iRetVal < 0){
        close( iFd ) ;
        return( iRetVal ) ;
      }
      if( eSockMode == udpWrite){
        if(setsockopt( iFd, IPPROTO_IP, IP_MULTICAST_IF,
               &inAddrIF, sizeof(inAddrIF) ) < 0 ) {
          perror( "setsockopt() IPPROTO_IP, IP_MULTICAST_IF" ) ;
          close( iFd ) ;
          return( -1 ) ;
        }
      }
    }

    switch(eSockMode){
```

```
  case udpRead:
  case udpReadWrite:
    iUtil = 1;
    if(setsockopt(iFd, SOL_SOCKET, SO_REUSEADDR, &iUtil, sizeof(iUtil))
       < 0 ) {
      close( iFd ) ;
      perror( "setsockopt() SOL_SOCKET SO_REUSEADDR" ) ;
      return( -1 ) ;
    }
#ifdef SO_REUSEPORT
    if(setsockopt(iFd, SOL_SOCKET, SO_REUSEPORT, &iUtil, sizeof(iUtil))
       < 0 ) {
      close( iFd ) ;
      perror( "setsockopt() SOL_SOCKET SO_REUSEPORT" ) ;
      return( -1 ) ;
    }
#endif              /* SO_REUSEPORT */
    if (bind(iFd, pSockAddr, sizeof(*pSockAddr)) < 0){
      perror("bind()");
      close(iFd);
      return(-1);
    }
    if(sService == NULL){
      iLen = sizeof(*pSockAddr);
      if (getsockname(iFd, pSockAddr, &iLen) < 0){
        perror("getsockname()");
        close(iFd);
        return(-1);
      }
    }
#ifdef WANT_STRUCT_ASSIGN
    ipMRequest.imr_multiaddr = inAddrGrp; /*struct assign*/
    ipMRequest.imr_interface = inAddrIF;  /*struct assign*/
#endif /* WANT_STRUCT_ASSIGN */
    memcpy(&ipMRequest.imr_multiaddr, &inAddrGrp, sizeof(inAddrGrp));
    memcpy(&ipMRequest.imr_interface, &inAddrIF, sizeof(inAddrIF));
    if (setsockopt(iFd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &ipMRequest,
           sizeof(ipMRequest)) < 0){
      perror("setsockopt() IPPROTO_IP IP_ADD_MEMBERSHIP");
      close(iFd);
      return(-1);
    }
    if(eSockMode == udpRead){
      break;
    }
    /*fall-thru for udpReadWrite */
  case udpWrite:
    pSockAddr->sin_addr.s_addr = inAddrGrp.s_addr; /*send to group*/
    cUtil = fLoopBack;
    if (setsockopt(iFd, IPPROTO_IP, IP_MULTICAST_LOOP, &cUtil,
           sizeof(cUtil)) < 0){
      perror("setsockopt IPPROTO_IP IP_MULTICAST_LOOP");
      close(iFd);
      return(-1);
    }

    if ((iTTL <= 0) || (iTTL > SHASTRA_MAX_TTL)){
      cUtil = SHASTRA_DEF_TTL;
    }
    else{
      cUtil = iTTL;
    }
    if (setsockopt(iFd, IPPROTO_IP, IP_MULTICAST_TTL, &cUtil,
           sizeof(cUtil)) < 0){
      perror("setsockopt IPPROTO_IP IP_MULTICAST_TTL");
      close(iFd);
      return(-1);
    }
    break;
  }

#ifdef WANT_FIONBIO
  cUtil = 1;
  if (ioctl(iFd, FIONBIO, &cUtil) < 0){
    perror("ioctl() FIONBIO");
    close(iFd);
    return(-1);
  }
#else               /*WANT_FIONBIO*/
  if( fcntl( iFd, F_SETFL, FNDELAY ) < 0 ) {
    perror( "fcntl() F_SETFL FNDELAY" ) ;
    close(iFd);
    return( -1 ) ;
  }
#endif              /*WANT_FIONBIO*/

  return(iFd);
#else               /*HAVEMULTICAST*/
  return -1;
#endif              /*HAVEMULTICAST*/
}

/*
 * getMyHostInAddr()-- Get my own host internet address
 */

int
cmGetMyHostInAddr(psaInHost)
     struct sockaddr_in *psaInHost;
{
  char sbHost[256];
  struct hostent *pheHost;

#ifdef SHASTRA4SUN5
  if (sysinfo(SI_HOSTNAME,sbHost, sizeof(sbHost)) < 0){
    fprintf(stderr,"sysinfo()-> Unknown Host Name!\n");
    return(-1);
  }
#else
  if (gethostname(sbHost, sizeof(sbHost)) < 0){
    fprintf(stderr,"gethostname()-> Unknown Host Name!\n");
    return(-1);
  }
#endif
  pheHost = gethostbyname(sbHost);
  if (!pheHost){
    fprintf(stderr,"gethostbyname()-> Unknown Host %s\n", sbHost);
    return(-1);
  }
  psaInHost->sin_family = AF_INET;
  psaInHost->sin_port = 0;
  memcpy(&psaInHost->sin_addr, pheHost->h_addr, sizeof(psaInHost->sin_addr)
      );

  fprintf(stderr,"Host %s, Address:%ld (0x%lx)\n",
      sbHost, psaInHost->sin_addr.s_addr, psaInHost->sin_addr.s_addr);
  return(0);
}


/*
 * sendUDPPacket()--
 */

int
cmSendUDPPacket(iFd, sMessage, lMessage, pSockAddr)
     int            iFd;
     char *        sMessage;
     int            lMessage;
     struct sockaddr_in *pSockAddr;
{
  int retVal;

  retVal = sendto(iFd, sMessage, lMessage, 0, pSockAddr, sizeof(*pSockAddr)
      );
  if(retVal < 0){
    perror("sendto()");
    return -1;
  }
  return retVal;
}

/*
 *  recvUDPPacket()--
 */
int
cmRecvUDPPacket(iFd, sMessage, lMaxLen, fIgnoreOwn)
     int iFd;
     char *sMessage;
     int lMaxLen;
     enum udpPacketMode fIgnoreOwn;
{
  struct sockaddr_in pFromAddr;
  int lAddr = sizeof(pFromAddr);
  int lMessage;

  do{
    lMessage = recvfrom(iFd, sMessage, lMaxLen, 0, &pFromAddr, &lAddr);
    fprintf(stderr,"cmRecvUDPPacket()->   ");
    if (lMessage < 0){
      if (errno == EWOULDBLOCK)
        return(0);
      else{
        perror("cmRecvUDPPacket()->recvfrom()");
        exit(-1);
      }
    }
    if (lMessage == 0){
      break;
    }
  } while ((fIgnoreOwn == udpIgnoreOwn) &&
       (pFromAddr.sin_addr.s_addr == saInMine.sin_addr.s_addr));

  return(lMessage);
}


#ifdef STANDALONE
int
cmUdpRecvHandler(iFd)
     int iFd;
{
  char sbBuffer[256];
  int lMessage;

  lMessage = cmRecvUDPPacket(iFd, sbBuffer, 256, udpAcceptOwn);
  fprintf(stdout, "cmUdpRecvHandler()->recv'd %d (%s)\n", lMessage,
      sbBuffer);
}

int
cmUdpSendHandler(iFd)
     int iFd;
{
  extern struct sockaddr_in sockAddr;
  extern int myFD;
  struct sockaddr_in *pSockAddr = &sockAddr;
  char sbBuffer[256], *sInput;
  int lMessage, lSent;

  sInput = fgets(sbBuffer, 256, stdin);
  if(sInput == NULL){
    exit(0);
  }
  lMessage = strlen(sInput);
  sbBuffer[lMessage - 1] = '\0';
  lSent = cmSendUDPPacket(myFD, sbBuffer, lMessage, pSockAddr);
  fprintf(stderr, "cmUdpSendHandler()->sent %d of %d (%s)\n",
      lSent, lMessage, sbBuffer);

}

enum udpCommMode eUDPMode = udpMulticast;  /* default multicast */
int myFD;
struct sockaddr_in sockAddr;

int
main(argc, argv)
     int argc;
     char **argv;
{
  int cmUdpRecvHandler(), cmUdpSendHandler();

  (void) cmGetMyHostInAddr(&saInMine);

  switch(eUDPMode){
  case udpMulticast:
    myFD = cmSetupMCastSocket(SHASTRA_MCAST_SERVICE, SHASTRA_GUESS_PORT,
                NULL, NULL, SHASTRA_MCAST_ADDR,
                SHASTRA_DEF_TTL, TRUE, udpReadWrite, &sockAddr);
    break;
  case udpBroadcast:
    myFD = cmSetupBCastSocket(SHASTRA_BCAST_SERVICE, SHASTRA_GUESS_PORT,
                NULL, NULL, udpReadWrite, &sockAddr);
    break;
  default:
  case udpUnicast:
    myFD = cmSetupUCastSocket(SHASTRA_UCAST_SERVICE, SHASTRA_GUESS_PORT,
                udpReadWrite, &sockAddr);
    break;
  }
  if(myFD < 0){
    fprintf(stderr,"main()->couldn't set up socket for %s!\n",
        (eUDPMode == udpMulticast)?"MULTICAST":
        (eUDPMode == udpBroadcast)?"BROADCAST":"UNICAST");
    exit(-1);
  }

  mplexInit(NULL, NULL);
  if (mplexRegisterChannel(myFD, cmUdpRecvHandler, NULL, NULL) < 0 ) {
    fprintf(stderr, "main()->Couldn't register Recv Handler!\n");
  }
  if (mplexRegisterChannel(0, cmUdpSendHandler, NULL, NULL) < 0) {
    fprintf(stderr, "main()->Couldn't register Send Handler!\n");
  }
  cmShowInterfaces(myFD);

  mplexMain(NULL);

}

#endif /*STANDALONE*/
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <pwd.h>
#ifdef SHASTRA4SUN5
#include <sys/systeminfo.h>
char *strdup(char *);
int putenv(char *);
#endif

#include <sys/errno.h>
#include <netdb.h>

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Xutil.h>

#include <Xm/Text.h>

#include <shastra/shastra.h>
#include <shastra/shastraStateDefs.h>

#include <shastra/utils/list.h>

#include <shastra/uitools/strListUtilities.h>
#include <shastra/uitools/chooseOne.h>
#include <shastra/uitools/chooseMany.h>
```

```
#include <shastra/uitools/confirmCB.h>

#include <shastra/network/server.h>
#include <shastra/network/mplex.h>
#include <shastra/network/hostMgr.h>
#include <shastra/network/sharedMem.h>

#include <shastra/datacomm/shastraIdH.h>
#include <shastra/datacomm/shastraIdTagH.h>

#include <shastra/shautils/shautils.h>
#include <shastra/shautils/kernelFronts.h>
#include <shastra/shautils/sesMgrFrontsP.h>
#include <shastra/shautils/sesMgrFronts.h>

#include <shastra/kernel/kernel_server.h>

#include <shastra/session/sesMgr.h>
#include <shastra/session/sesMgrMainCB.h>
#include <shastra/session/sesMgr_server.h>
#include <shastra/session/sesMgr_client.h>
#include <shastra/session/sesMgrState.h>

static char *GetShastraBaseDir();
int getCmdLineArgs(Prot2(int, char **));
static shaSesMgrAppData sesMgrAppData;
shaSesMgrAppData *pSesMgrAppData = &sesMgrAppData;
static shastraId sesMgrShastraId;
shastraId       *pSesMgrSId = &sesMgrShastraId;

shastraIdTags   sesMgrStartIdTags;
shastraIdTags   sesMgrStartPermTags;
collabData     *pSesMgrCollData;
char            sbOutMsgBuf[1024];
#define DEBUG 0
int             debug = DEBUG;
extern int      errno;

int             kernelPortNum;
int             mainKernClntSocket;
unsigned long   kernelIPAddr;
int             iKernelFrontIndex;
int             iSesMgrFrontIndex;
#ifndef SHASTRA4SUN5
#define MAXNAMELEN 128
#endif
char            kernelHostName[MAXNAMELEN];
char            kernelUserName[MAXNAMELEN];
char            kernelHeadHostName[MAXNAMELEN];

shastraId       kernelShastraId;
shastraIds     *pShastraFrontIds;   /* fronts connected on kernel */
shastraIdTags  *pShastraFrontIdTags;    /* fronts connected on kernel */
shastraIdTags  *pShastraFrontPermTags;  /* fronts connected on kernel */
sesmFronts     *pSesmFrontCD;

int             shastraServerStatus;

char           *shastraPasswd = SHASTRAPASSWORD;

char           *kernelAppName;
char           *kernelDispName;
char           *kernelPasswd;
char           *kernelCollType;
unsigned long   kernelPerms;
unsigned long   kernelIdTag;
int             kernelFNoGUI;
int             kernelFAutoJoin;

shaCmdData      serverCmdData;
cmCommand       serverCommandTab[] = SESMGRCMDS;
#define NSESMGRCMDS (sizeof(serverCommandTab)/sizeof(cmCommand))
/* number of commands */
int             serverNCmds = NSESMGRCMDS;

void            (*collabTerminateFunc) ();
void            (*collabJoinFunc) ();
void            (*collabLeaveFunc) ();
void            (*collabRemoveFunc) ();

int             shastraServiceSocket;

shaCmdData      kernelCmdData;

cmCommand       kernelCmdTab[] = SESMGR_CLIENTCMDS;
#define SESMGR_NCMDS (sizeof(kernelCmdTab)/sizeof(cmCommand))
int             kernelNCmds = SESMGR_NCMDS;

cmCommand       kernelInCmdTab[] = SESMGR_CLIENTINCMDS;
#define SESMGR_INNCMDS (sizeof(kernelInCmdTab)/sizeof(cmCommand))
int             kernelInNCmds = SESMGR_INNCMDS;

hostData        hostMainKern;
hostData       *pHostMainKern = &hostMainKern;

void
shastraSesMgrSetupApplResDir()
{
  char sbName[1024], *sName;

  sName = resolveNameFromBase(pSesMgrAppData->sDirBase,
                pSesMgrAppData->sDirDefs);
    sprintf(sbName,"XAPPLRESDIR=%s", sName);
    putenv(sbName);
}
```

```
Widget
shastraSmMain(argc, argv, sSMName, wgParent, pCollCmdData)
    int             argc;
    char          **argv;
    char *sSMName;
    Widget          wgParent;
    shaCmdData     *pCollCmdData;
{
    char *sName;
    struct hostent *pHostEnt;
    int             i;
    Widget          wgMainCmdShell;
    extern int      closedChannelCleanUpHandler();
        uid_t auid;
        struct passwd *apass;
        unsigned int itemp;

    static XtResource xrmResources[] = {
     { XshaNbaseDirectory, XshaCbaseDirectory, XtRString, sizeof(String),
      XtOffsetOf(shaSesMgrAppData, sDirBase), XtRImmediate,
      (XtPointer)DEFSHASTRABASEDIR },
     { XshaNminimal, XshaCminimal, XtRBoolean, sizeof(Boolean),
      XtOffsetOf(shaSesMgrAppData, fMinimal), XtRImmediate, (XtPointer)
          False },
     { XshaNconnect, XshaCconnect, XtRBoolean, sizeof(Boolean),
      XtOffsetOf(shaSesMgrAppData, fConnect), XtRImmediate, (XtPointer)True
          },
     { XshaNnoGUI, XshaCnoGUI, XtRBoolean, sizeof(Boolean),
      XtOffsetOf(shaSesMgrAppData, fNoGUI), XtRImmediate, (XtPointer)False
          },
     { XshaNusePixmap, XshaCusePixmap, XtRBoolean, sizeof(Boolean),
      XtOffsetOf(shaSesMgrAppData, fPixmap), XtRImmediate, (XtPointer)False
          },
     { XshaNhelp, XshaChelp, XtRBoolean, sizeof(Boolean),
      XtOffsetOf(shaSesMgrAppData, fHelp), XtRImmediate, (XtPointer)False }
          ,
     { XshaNservicePort, XshaCservicePort, XtRInt, sizeof(int),
      XtOffsetOf(shaSesMgrAppData, iSvcPort), XtRImmediate, (XtPointer)0 },
     { XshaNshastraPort, XshaCshastraPort, XtRInt, sizeof(int),
      XtOffsetOf(shaSesMgrAppData, iShaPort), XtRImmediate, (XtPointer)0 },
     { XshaNdebugLevel, XshaCdebugLevel, XtRInt, sizeof(int),
      XtOffsetOf(shaSesMgrAppData, iDbgLevel), XtRImmediate, (XtPointer)0 }
          ,
     { XshaNdefsDirectory, XshaCdefsDirectory, XtRString, sizeof(String),
      XtOffsetOf(shaSesMgrAppData, sDirDefs), XtRImmediate,
      (XtPointer)DEFSHASTRADEFSDIR },
     { XshaNdataDirectory, XshaCdataDirectory, XtRString, sizeof(String),
      XtOffsetOf(shaSesMgrAppData, sDirData), XtRImmediate,
      (XtPointer)DEFSHASTRADATADIR },
     { XshaNbinDirectory, XshaCbinDirectory, XtRString, sizeof(String),
      XtOffsetOf(shaSesMgrAppData, sDirBin), XtRImmediate,
      (XtPointer)DEFSHASTRABINDIR },
     { XshaNlogFile, XshaClogFile, XtRString, sizeof(String),
      XtOffsetOf(shaSesMgrAppData, sFileLog), XtRImmediate,
      (XtPointer)DEFSHASTRALOGFILE },
     { XshaNhomeFile, XshaChomeFile, XtRString, sizeof(String),
      XtOffsetOf(shaSesMgrAppData, sFileHome), XtRImmediate,
      (XtPointer)DEFSHASTRAHOMEFILE },
     { XshaNappsFile, XshaCappsFile, XtRString, sizeof(String),
      XtOffsetOf(shaSesMgrAppData, sFileApps), XtRImmediate,
      (XtPointer)DEFSHASTRAAPPSFILE },
     { XshaNusersFile, XshaCusersFile, XtRString, sizeof(String),
      XtOffsetOf(shaSesMgrAppData, sFileUsers), XtRImmediate,
      (XtPointer)DEFSHASTRAUSERSFILE },
     { XshaNhostsFile, XshaChostsFile, XtRString, sizeof(String),
      XtOffsetOf(shaSesMgrAppData, sFileHosts), XtRImmediate,
      (XtPointer)DEFSHASTRAHOSTSFILE },
     { XshaNlocalStarter, XshaClocalStarter, XtRString, sizeof(String),
      XtOffsetOf(shaSesMgrAppData, sLocStart), XtRImmediate,
      (XtPointer)DEFSHASTRASTARTLOCAL },
     { XshaNremoteStarter, XshaCremoteStarter, XtRString, sizeof(String),
      XtOffsetOf(shaSesMgrAppData, sRemStart), XtRImmediate,
      (XtPointer)DEFSHASTRASTARTREMOTE },
     { XshaNpassword, XshaCpassword, XtRString, sizeof(String),
      XtOffsetOf(shaSesMgrAppData, sPasswd), XtRImmediate,
      (XtPointer)DEFSHASTRAPASSWD },
    };

    xrmResources[0].default_addr = GetShastraBaseDir();
    XtVaGetApplicationResources(wgParent,
        (XtPointer)&sesMgrAppData,
        xrmResources, XtNumber(xrmResources),
        /*hardcoded non-overridable app resources vararg list*/
        XshaNhelp, False,
        XshaNusePixmap, False,
        NULL);
    /*sanity checking of resources*/

/*
    shastraSesMgrSetupApplResDir();
*/
    pSesMgrAppData->sName = sSMName;
    getCmdLineArgs(argc, argv);
    kernelAppName = pSesMgrAppData->sName;/* store application name */
    if (kernelDispName == NULL) {
        kernelDispName = XDisplayName(NULL);
    }
    if (kernelPasswd == NULL) {
        kernelPasswd = SHASTRAPASSWORD;
    }
    registerInit();
    kernFrontsInit();
    sesmFrontsInit();
    mplexRegisterErrHandler(closedChannelCleanUpHandler);
```

```
#ifdef SHASTRA4SUN5
    if (sysinfo(SI_HOSTNAME,kernelHostName, MAXNAMELEN) < 0) {
        perror("sysinfo()");
        strcpy(kernelHostName, "anonymous.cs.purdue.edu");
    }
#else
    if (gethostname(kernelHostName, MAXNAMELEN) != 0) {
        perror("gethostname()");
        strcpy(kernelHostName, "anonymous.cs.purdue.edu");
    }

#endif
    if ((pHostEnt = gethostbyname(kernelHostName)) == NULL) {
        perror("gethostbyname()");
        return 0;
    }
    memcpy(&itemp, pHostEnt->h_addr_list[0], sizeof(unsigned int));
        kernelIPAddr = ntohl(itemp);
    /*kernelIPAddr = *(unsigned long *) &pHostEnt->h_addr_list[0][0];*/
        auid = getuid();
        apass = getpwuid(auid);
        strcpy(kernelUserName,apass->pw_name);
    /*
     * printf("name : %s\n",kernelHostName);
     */
    serverCmdData.pCmdTab = serverCommandTab;
    serverCmdData.nCmds = serverNCmds;
    serverCmdData.pCmdTabIn = NULL;
    serverCmdData.nCmdsIn = 0;

    if ((kernelPortNum = cmOpenServerSocket(TESTSESM_SERVICE_NAME, 0,
              &serverCmdData, &shastraServiceSocket, NULL)) == -1) {
        /* OpenServerSocket registers the handler */
        fprintf(stderr, "main()->Server Start-up error!\n Quitting!\n");
        exit(-1);
    }
    cmJoinCmdData(&serverCmdData, pCollCmdData);
    /* add sesm-specific commands to table */

    getRegisterInfo(&kernelShastraId);

    wgMainCmdShell = createMainCmdShell(wgParent);

    /* connect to kernel */

    for (i = 0; i < 3; i++) {   /* max 3 tries */
        shastraServerStatus = cmClientConnect2Server(kernelHostName,
                  SHASTRA_SERVICE_NAME, 0, &mainKernClntSocket);
        if ((shastraServerStatus == -1) && (errno == ECONNREFUSED)) {
            /* problem.. maybe no kernel */
            sName = resolveNameFrom2Bases(pSesMgrAppData->sDirBase,
                pSesMgrAppData->sDirBin, pSesMgrAppData->sLocStart);
            startShastraKernel(&kernelShastraId, sName);
            } else {
                break;
            }
        }
        if (shastraServerStatus == -1) {
            fprintf(stderr, "main()--No Server..Quitting!!\n");
            exit(-1);
        }

        kernelCmdData.pCmdTab = kernelCmdTab;
        kernelCmdData.nCmds = kernelNCmds;
        kernelCmdData.pCmdTabIn = kernelInCmdTab;
        kernelCmdData.nCmdsIn = kernelInNCmds;

        pHostMainKern->fdSocket = mainKernClntSocket;
        pHostMainKern->sendList = listMakeNew();
        pHostMainKern->recvList = listMakeNew();
        pHostMainKern->fStatus = shaWait2Send;

        /* register handler */
        if (mplexRegisterChannel(pHostMainKern->fdSocket, shaClientHandler,
                     &kernelCmdData, NULL) == -1) {
            fprintf(stderr, "main()->Couldn't Register Client Handler!!\n");
            pHostMainKern->fStatus = shaError;
            return(0);
        }
        mplexSetHostData(pHostMainKern->fdSocket, pHostMainKern);
        /* after connecting,setting up handler */
        setShaSesmIdOprn(0);    /* register ID with MainKernel */
        /* NOW invite collab participants */
    fprintf(stderr, "in session manager!\n");
        if (sesMgrStartIdTags.shastraIdTags_len > 0) {
            collStartTellJoinOprn(0);
            for (i = 1; i < sesMgrStartIdTags.shastraIdTags_len; i++) {
                /* not from 0; 0 is chief of collab */
                if(kernelFAutoJoin){
                    collStartTellJoinOprn(i);
                }
                else{
                    collStartInviteJoinOprn(i);
                }
            }
        }
        /* identify front index */
        iSesMgrFrontIndex =
            locateSesmFronts((shastraIdTag *) & kernelShastraId.lSIDTag);
        if (iSesMgrFrontIndex != -1) {
            fprintf(stderr, "main()->locateSesmFronts() already has index %d!\
                n",
                iSesMgrFrontIndex);
        } else {
            iSesMgrFrontIndex = occupySmFrFreeSlot(
                    (shastraIdTag *) & kernelShastraId.lSIDTag);
    }
    pSesmFrontCD = getSesMgrCntlData((shastraIdTag *)& kernelShastraId.
        lSIDTag);
    pShastraFrontIdTags = getSesmFrontSIdTags((shastraIdTag *)
        & kernelShastraId.lSIDTag);
    pShastraFrontPermTags = getSesmFrontPermTags((shastraIdTag *)
        & kernelShastraId.lSIDTag);
    pSesMgrCollData = (collabData *) malloc(sizeof(collabData));
    memset(pSesMgrCollData, 0, sizeof(collabData));
    pSesMgrCollData->pShmInfoOut = shmInfoCreate();
    if (setSesMgrData((shastraIdTag *) & kernelShastraId.lSIDTag,
              (char *) pSesMgrCollData) < 0) {
        fprintf(stderr, "main()->couldn't setSesMgrData!\n");
    }
    iKernelFrontIndex = locateKernFronts(&kernelShastraId);
    if (iKernelFrontIndex != -1) {
        fprintf(stderr, "main()->locateKernFronts() already has index %d!\
            n",
            iKernelFrontIndex);
    } else {
        iKernelFrontIndex = occupyKrFrFreeSlot(&kernelShastraId);
    }
    pShastraFrontIds = getKernFrontSIds(&kernelShastraId);
    /* initially empty fronts */
    pShastraFrontIds->shastraIds_len = 0;
    pShastraFrontIds->shastraIds_val =
        (shastraId_P *) malloc(mplexGetMaxChannels() * sizeof(shastraId_P))
            ;
    pShastraFrontIds = (shastraIds *)malloc(sizeof(shastraIds));
    pShastraFrontIds->shastraIds_len = 0;
    pShastraFrontIds->shastraIds_val =
        (shastraId_P *) malloc(mplexGetMaxChannels() * sizeof(shastraId_P))
            ;

    if (rgsbShastraFront != NULL) {
        strListDestroy(rgsbShastraFront);
    }
    rgsbShastraFront = pSIds2StrTab(pShastraFrontIds, PSIDNMHOST |
        PSIDNMAPPL);
    chooseOneChangeList(pcoShastraFront, rgsbShastraFront,
                coNoInitialHighlight);

    return( wgMainCmdShell);
}



int
getRegisterInfo(pSId)
    shastraId      *pSId;
{

    pSId->lIPAddr = kernelIPAddr;
    printf("%lu (%lx) -- %s\n", pSId->lIPAddr, pSId->lIPAddr,
           ipaddr2str(pSId->lIPAddr));

    pSId->lSIDTag = (kernelIPAddr << 16) + getpid();
    /* for sesMgrs pid+IPAddr is thier tag */

    pSId->dLoadAvg = 0;

    pSId->nmHost = strdup(kernelHostName);
    pSId->nmDisplay = strdup(kernelDispName);
    pSId->nmApplicn = strdup(kernelAppName);
    pSId->nmUser = strdup(kernelUserName);
    pSId->webname = strdup(kernelUserName);
    pSId->nmPasswd = strdup(kernelPasswd);

    pSId->iPort = kernelPortNum;

    pSId->iProcId = getpid();

    if (debug) {
        outputId(stdout, pSId);
    }
        return(0);
}

/*
 * Function --
 */
void
showInfo(s)
    char            *s;
{
    static XmTextPosition currentPosn;
    outputTextToWidget(s, wgStatusText, &currentPosn);
    /*
     * fprintf(stdout, "%s", s);
     */
}

int
cmdLineUsage(argv)
    char            **argv;
{
    fprintf(stderr, "usage: %s [options]\n", argv[0]);
    fprintf(stderr, "  where options are:\n");
    fprintf(stderr, "    -display <display name>\n");
    fprintf(stderr, "    -help\n");
    fprintf(stderr, "    -nogui\n");
    fprintf(stderr, "    -passwd <password>\n");
    exit(1);
}

int
```

```
getCmdLineArgs(argc, argv)
    int             argc;
    char           **argv;
{
    int             i;
    int             j;

    /* allocate space for cmdline arg tags */
    kernelPerms = 0 |
        SHASTRA_PERM_ACCESS |
        SHASTRA_PERM_BROWSE |
        SHASTRA_PERM_MODIFY;
    sesMgrStartIdTags.shastraIdTags_len = 0;
    sesMgrStartIdTags.shastraIdTags_val = (shastraIdTag *) malloc(
                 sizeof(shastraIdTag) * mplexGetMaxChannels());
    memset(sesMgrStartIdTags.shastraIdTags_val,0,
          sizeof(shastraIdTag) * mplexGetMaxChannels());
    sesMgrStartPermTags.shastraIdTags_len = 0;
    sesMgrStartPermTags.shastraIdTags_val = (shastraIdTag *) malloc(
                 sizeof(shastraIdTag) * mplexGetMaxChannels());
    memset(sesMgrStartPermTags.shastraIdTags_val,0,
          sizeof(shastraIdTag) * mplexGetMaxChannels());

    for (i = 1; i < argc; i++) {
        if (!strcmp("-display", argv[i])) {
            if (++i >= argc)
                cmdLineUsage(argv);
            kernelDispName = argv[i];
            continue;
        }
        if (!strcmp("-help", argv[i])) {
            cmdLineUsage(argv);
        }
        if (!strcmp("-nogui", argv[i])) {
            kernelFNoGUI = 1;
            continue;
        }
        if (!strcmp("-auto", argv[i])) {
            kernelFAutoJoin = 1;
            continue;
        }
        if (!strcmp("-idtag", argv[i])) {
            if (++i >= argc)
                cmdLineUsage(argv);
            kernelIdTag = atoi(argv[i]);
            continue;
        }
        if (!strcmp("-perms", argv[i])) {
            if (++i >= argc)
                cmdLineUsage(argv);
            kernelPerms = atoi(argv[i]);
            continue;
        }
        if (!strcmp("-passwd", argv[i])) {
            if (++i >= argc)
                cmdLineUsage(argv);
            kernelPasswd = argv[i];
            continue;
        }
        if (!strcmp("-tags", argv[i])) {
            for (j = 0; argc > (i + j + 1); j++) {
                /*
                 * will fail for negative tags!!. tags
                 * shouldn't be negative
                 */
                if (*argv[i + j + 1] != '-') {
                    sscanf(argv[i + j + 1], "%lu",
                           &sesMgrStartIdTags.shastraIdTags_val[j]);
                } else {
                    break;
                }
            }
            sesMgrStartIdTags.shastraIdTags_len = j;
            sesMgrStartIdTags.shastraIdTags_val = (shastraIdTag *) realloc(
                    sesMgrStartIdTags.shastraIdTags_val,
                          sizeof(shastraIdTag) * j);
            if (debug) {
                outputIdTags(stderr, &sesMgrStartIdTags);
            }
            i = i + j;
            continue;
        }
        if (!strcmp("-type", argv[i])) {
            if (++i >= argc)
                cmdLineUsage(argv);
            kernelCollType = argv[i];
            continue;
        }
        cmdLineUsage(argv);
    }

    sesMgrStartPermTags.shastraIdTags_len =
        sesMgrStartIdTags.shastraIdTags_len;
    sesMgrStartPermTags.shastraIdTags_val[0] = kernelPerms |
        (SHASTRA_PERM_GRANT | SHASTRA_PERM_COPY);
    for (i = 1; i < sesMgrStartIdTags.shastraIdTags_len; i++) {
        sesMgrStartPermTags.shastraIdTags_val[i] = kernelPerms;
    }
    sesMgrStartPermTags.shastraIdTags_val = (shastraIdTag *) realloc(
                      sesMgrStartPermTags.shastraIdTags_val,
          sizeof(shastraIdTag) * sesMgrStartPermTags.shastraIdTags_len);
        return(0);
}

void
registerCollabTerminateFunc(func)
```

```
    void            (*func) ();
{
    collabTerminateFunc = func;
}

void
registerCollabJoinFunc(func)
    void            (*func) ();
{
    collabJoinFunc = func;
}

void
registerCollabLeaveFunc(func)
    void            (*func) ();
{
    collabLeaveFunc = func;
}

void
registerCollabRemoveFunc(func)
    void            (*func) ();
{
    collabRemoveFunc = func;
}


shastraId *
getMySesMgrShastraId()
{
    if(pSesMgrAppData){
        return pSesMgrAppData->pSIdSelf;
    }
    else{
        return NULL;
    }
}

shaSesMgrAppData *
getMySesMgrAppData()
{
    return pSesMgrAppData;
}

static char *GetShastraBaseDir()
{
    char *dname;

    if (dname = getenv("SHASTRADIR"))
    {
         return(dname);
    }
    else
```

```
        {
            dname = strdup(DEFSHASTRABASEDIR);
        }
        return(dname);
    }
```

```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
#include <stdio.h>
#include <sys/errno.h>

#include <shastra/utils/list.h>

#include <shastra/uitools/chooseOne.h>
#include <shastra/uitools/strListUtilities.h>
#include <shastra/uitools/callbackArg.h>

#include <shastra/network/server.h>
#include <shastra/network/mplex.h>
#include <shastra/network/hostMgr.h>

#include <shastra/datacomm/shastraIdH.h>
#include <shastra/datacomm/shastraIdTagH.h>
#include <shastra/datacomm/shastraDataH.h>

#include <shastra/shautils/shautils.h>
#include <shastra/shautils/kernelFronts.h>
#include <shastra/shautils/sesMgrFronts.h>

#include <shastra/kernel/kernel_server.h>

#include <shastra/session/sesMgr.h>
#include <shastra/session/sesMgr_client.h>

#define checkConn()         \
    if (pHostMainKern->fStatus == shaError) {                \
        fprintf(stderr,"Connection to Shastra is bad!\n");  \
            return; \
        }

    #define sendReqString(s, arg)       \
        if(hostSendQueuedRequest(pHostMainKern, s, arg) == -1){ \
            pHostMainKern->fStatus = shaError;      \
            fprintf(stderr,"Error in Sending Shastra Operation Request\n"); \
            return; \
        }


    #define ShastraIdIn(filedesc, pShaId)            \
        if(shastraIdIn(pHostMainKern->fdSocket, pShaId) == -1){ \
            pHostMainKern->fStatus = shaError;\
            closedChannelCleanUpHandler(pHostMainKern->fdSocket);\
            fprintf(stderr, "Error Receiving SID from Kernel\n");   \
            return;\
        }

    #define ShastraIdOut(filedesc, pShaId)           \
        if(shastraIdOut(pHostMainKern->fdSocket, pShaId) == -1){    \
            pHostMainKern->fStatus = shaError;\
            closedChannelCleanUpHandler(pHostMainKern->fdSocket);\
            fprintf(stderr, "Error Sending SID to Kernel\n");   \
            return; \
        }

    #define ShastraIdsIn(filedesc, pShaIds)          \
        if(shastraIdsIn(pHostMainKern->fdSocket, pShaIds) == -1){   \
            pHostMainKern->fStatus = shaError;\
            closedChannelCleanUpHandler(pHostMainKern->fdSocket);\
            fprintf(stderr, "Error Receiving SIDs from Kernel\n");  \
            return; \
        }

    #define ShastraIdsOut(filedesc, pShaIds)             \
        if(shastraIdsOut(pHostMainKern->fdSocket, pShaIds) == -1){  \
            pHostMainKern->fStatus = shaError;\
            closedChannelCleanUpHandler(pHostMainKern->fdSocket);\
            fprintf(stderr, "Error Sending SIDs to Kernel\n");  \
            return; \
        }

    #define ShastraIdTagIn(filedesc, pShaIdTag)          \
        if(shastraIdTagIn(pHostMainKern->fdSocket, pShaIdTag) == -1){   \
            pHostMainKern->fStatus = shaError;\
            closedChannelCleanUpHandler(pHostMainKern->fdSocket);\
            fprintf(stderr, "Error Receiving SIDTag from Kernel\n");\
            return; \
        }

    #define ShastraIdTagOut(filedesc, pShaIdTag)             \
        if(shastraIdTagOut(pHostMainKern->fdSocket, pShaIdTag) == -1){  \
        pHostMainKern->fStatus = shaError;\
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);\
        fprintf(stderr, "Error Sending SIDTag to Kernel\n");    \
        return;\
    }

#define ShastraIdTagsIn(filedesc, pShaIdTags)           \
    if(shastraIdTagsIn(pHostMainKern->fdSocket, pShaIdTags) == -1){ \
        pHostMainKern->fStatus = shaError;\
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);\
        fprintf(stderr, "Error Receiving SIDTags from Kernel\n");\
        return;\
    }

#define ShastraIdTagsOut(filedesc, pShaIdTags)          \
    if(shastraIdTagsOut(pHostMainKern->fdSocket, pShaIdTags) == -1){\
        pHostMainKern->fStatus = shaError;\
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);\
        fprintf(stderr, "Error Sending SIDTags to Kernel\n");   \
        return; \
    }

#define ShastraULongIn(filedesc, pULong)            \
    if(shaULongIn(pHostMainKern->fdSocket, pULong) == -1){    \
        pHostMainKern->fStatus = shaError;                          \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);
            \
        fprintf(stderr, "Error Receiving pULong from kernel\n");    \
        return;                                             \
    }

#define ShastraULongOut(filedesc, pULong)           \
    if(shaULongOut(pHostMainKern->fdSocket, pULong) == -1){   \
        pHostMainKern->fStatus = shaError;                          \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);
            \
        fprintf(stderr, "Error Sending pULong to Kernel\n");    \
        return;                                             \
    }



extern int      debug;

/*
 * Function
 */
void
endSystemOprn(iObjIndex)
    int             iObjIndex;
{
    shastraIds     *pSIds;
    shastraId      *pSId;
    pSIds = getKernFrontSIds(&kernelShastraId);
    pSId = pSIds->shastraIds_val[iObjIndex];
    if (debug) {
        outputId(stdout, pSId);
    }
    if (strcmp(pcbArgPopup->argBuffer, pSId->nmPasswd)) {
        /* passwd mismatch */
        sprintf(sbOutMsgBuf, "Kill()->Password Incorrect -- Aborted\n");
        showInfo(sbOutMsgBuf);
        return;
    }
    checkConn();
    sendReqString(REQ_END_SYSTEM, NULL);
    ShastraIdOut(pHostMainKern->fdSocket, pSId);
    cmFlush(pHostMainKern->fdSocket);
}

/*
 * Function
 */
void
setShaSesmIdOprn(i)
    int             i;
{

    checkConn();
    sendReqString(REQ_SET_SHASESMID, NULL);
    ShastraIdOut(pHostMainKern->fdSocket, &kernelShastraId);
    printf("%s\n", pSId2Str(&kernelShastraId, PSIDSHOWALL));
    cmFlush(pHostMainKern->fdSocket);
}

/*
 * Function
 */
void
setShaSesmFrIdOprn(i)
    int             i;
{
    checkConn();
    sendReqString(REQ_SET_SHASESMFRID, NULL);
    ShastraIdTagOut(pHostMainKern->fdSocket, & kernelShastraId.lSIDTag);
    ShastraIdTagsOut(pHostMainKern->fdSocket, pShastraFrontIdTags);
    ShastraIdTagsOut(pHostMainKern->fdSocket, pShastraFrontPermTags);   /*
        perms */
    cmFlush(pHostMainKern->fdSocket);
}

/*
 * Function
 */
void
```

```
getShaKernIdOprn(iObjIndex)
    int                iObjIndex;
{
    checkConn();
    sendReqString(REQ_GET_SHAKERNID, NULL);
    cmFlush(pHostMainKern->fdSocket);
}


/*
 * Function
 */
void
getShaKernFrIdOprn(iObjIndex)
    int                iObjIndex;
{
    shastraId      *pSId;

    checkConn();
    sendReqString(REQ_GET_SHAKERNFRID, NULL);
    pSId = shastraKernIds.shastraIds_val[iObjIndex];
    ShastraIdOut(pHostMainKern->fdSocket, pSId);
    cmFlush(pHostMainKern->fdSocket);

}

/*
 * Function
 */
void
getShaSesmIdOprn(iObjIndex)
    int                iObjIndex;
{
    checkConn();
    sendReqString(REQ_GET_SHASESMID, NULL);
    cmFlush(pHostMainKern->fdSocket);
}

/*
 * Function
 */
void
getShaSesmFrIdOprn(iObjIndex)
    int                iObjIndex;
{
    shastraIdTag   *pSIdTag;

    pSIdTag = & shastraSesmIds.shastraIds_val[iObjIndex]->lSIDTag;
    if (*pSIdTag == kernelShastraId.lSIDTag) {
        /* don't want to send request for myself */
        return;
    }
    checkConn();
```

```
        sendReqString(REQ_GET_SHASESMFRID, (char *) NULL);
        ShastraIdTagOut(pHostMainKern->fdSocket, pSIdTag);
        printf("%s\n", pSIdTag2Str(pSIdTag, 0));
        cmFlush(pHostMainKern->fdSocket);
    }

    /*
     * Function
     */
    void
    collStartInviteJoinOprn(iObjIndex)
        int             iObjIndex;
    {
        /* works off the start list */
        checkConn();
    fprintf(stderr, "Invite Join!\n");
        sendReqString(REQ_COLL_INVITEJOIN, NULL);
            ShastraIdTagOut(pHostMainKern->fdSocket, & kernelShastraId.lSIDTag)
                ;
        ShastraIdTagOut(pHostMainKern->fdSocket,
                &sesMgrStartIdTags.shastraIdTags_val[iObjIndex]);
        ShastraIdTagOut(pHostMainKern->fdSocket,
                &sesMgrStartIdTags.shastraIdTags_val[0]); /*leader*/
        ShastraIdTagOut(pHostMainKern->fdSocket,
                &sesMgrStartPermTags.shastraIdTags_val[iObjIndex]);
        cmFlush(pHostMainKern->fdSocket);
    }


    /*
     * Function
     */
    void
    collStartTellJoinOprn(iObjIndex)
        int             iObjIndex;
    {
        /* works off the start list */
        checkConn();
    fprintf(stderr, "IN session manager Sending: REQ_COLL_TELL_JOIN\n");
        sendReqString(REQ_COLL_TELLJOIN, NULL);
        ShastraIdTagOut(pHostMainKern->fdSocket, & kernelShastraId.lSIDTag);
        ShastraIdTagOut(pHostMainKern->fdSocket,
                &sesMgrStartIdTags.shastraIdTags_val[iObjIndex]);
        ShastraIdTagOut(pHostMainKern->fdSocket,
                &sesMgrStartPermTags.shastraIdTags_val[iObjIndex]);
        cmFlush(pHostMainKern->fdSocket);
    }


    /*
     * Function
     */
    void
```

```
collTellJoinOprn(pSmSIdTag, pSIdTag, pPermTag)
    shastraIdTag   *pSmSIdTag;
    shastraIdTag   *pSIdTag;
    shastraIdTag   *pPermTag;
{
    checkConn();
    sendReqString(REQ_COLL_TELLJOIN, NULL);
    ShastraIdTagOut(pHostKernel->fdSocket, pSmSIdTag);
    ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
    ShastraIdTagOut(pHostKernel->fdSocket, pPermTag);
    cmFlush(pHostMainKern->fdSocket);
}

/*
 * Function
 */
void
helpOprn(iObjIndex)
    int             iObjIndex;
{
    checkConn();
    sendReqString(REQ_HELP, NULL);
    cmFlush(pHostMainKern->fdSocket);
}

/*
 * Function
 */
void
quitOprn(iObjIndex)
    int             iObjIndex;
{
    extern collabData *pSesMgrCollData;

    if (pHostMainKern->fStatus != shaError) {
        sendReqString(REQ_QUIT, NULL);
        cmFlush(pHostMainKern->fdSocket);
    }
    shMemFree(pSesMgrCollData->pShmInfoOut);
    mplexUnRegisterChannel(pHostMainKern->fdSocket);
    exit(0);
}


/*
 * Function
 */
int
endSystemRespHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_END_SYSTEM);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
getShastraIdRespHandler(fd)
    int             fd;
{

    ShastraIdsIn(fd, &shastraSysIds);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHASTRAID);
    showInfo(sbOutMsgBuf);
    if (debug) {
        outputIds(stderr, &shastraSysIds);
    }
    if (rgsbShastraSys != NULL) {
        strListDestroy(rgsbShastraSys);
    }
    rgsbShastraSys = pSIds2StrTab(&shastraSysIds, PSIDSHOWALL);
    chooseOneChangeList(pcoShastraSys, rgsbShastraSys,
                coNoInitialHighlight);
}
/*
 * Function
 */
int
getShaKernIdRespHandler(fd)
    int             fd;
{

    ShastraIdsIn(fd, &shastraKernIds);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHAKERNID);
    showInfo(sbOutMsgBuf);
    if (debug) {
        outputIds(stderr, &shastraKernIds);
    }
    if (rgsbShastraKern != NULL) {
        strListDestroy(rgsbShastraKern);
    }
    rgsbShastraKern = pSIds2StrTab(&shastraKernIds, PSIDNMHOST);
    chooseOneChangeList(pcoShastraKern, rgsbShastraKern,
        coNoInitialHighlight);

    adjustKrFrMapSize(shastraKernIds.shastraIds_len);
    /* update map */
    updateKrFrMap(&shastraKernIds);
}

/*
 * Function
 */
int
```

```
getShaKernFrIdRespHandler(fd)
    int             fd;
{
    int             iObjIndex;
    static shastraId inShaId;
    static shastraIds inShaIds;
    shastraIds     *pSIds;
    int             krIndex;

    ShastraIdIn(fd, &inShaId);
    krIndex = locateKernFronts(&inShaId);
    if (krIndex == -1) {
        fprintf(stderr, "getShaKernFrIdRespHandler()->can't locate kernel\
            n");
        ShastraIdsIn(fd, &inShaIds);
        return -1;
    }
    pSIds = getKernFrontSIds(&inShaId);
    ShastraIdsIn(fd, pSIds);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHAKERNFRID);
    showInfo(sbOutMsgBuf);
    if (debug) {
        outputIds(stderr, pSIds);
    }
}


/*
 * Function
 */
int
getShaSesmIdRespHandler(fd)
    int             fd;
{

    ShastraIdsIn(fd, &shastraSesmIds);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHASESMID);
    showInfo(sbOutMsgBuf);
    if (debug) {
        outputIds(stderr, &shastraSesmIds);
    }
    if (rgsbShastraSesMgr != NULL) {
        strListDestroy(rgsbShastraSesMgr);
    }
    rgsbShastraSesMgr = pSIds2StrTab(&shastraSesmIds, PSIDNMHOST);
    chooseOneChangeList(pcoShastraSesMgr, rgsbShastraSesMgr,
            coNoInitialHighlight);
    adjustSmFrMapSize(shastraSesmIds.shastraIds_len);
    /* update map */
    updateSmFrMap(&shastraSesmIds);
}

/*
```

```
 * Function
 */
int
setShaSesmIdRespHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_SHASESMID);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
getShaSesmFrIdRespHandler(fd)
    int             fd;
{
    int             smIndex;
    static shastraIdTag inShaIdTag;
    static shastraIdTags inShaIdTags;
    shastraIdTags  *pSIdTags;
    shastraIdTags  *pPermTags;

    ShastraIdTagIn(fd, &inShaIdTag);
    if (inShaIdTag == kernelShastraId.lSIDTag) {
        /* don't want to accept info of myself */
        ShastraIdTagsIn(fd, &inShaIdTags);  /* tags */
        ShastraIdTagsIn(fd, &inShaIdTags);  /* perms */
        return 0;
    }
    smIndex = locateSesmFronts(&inShaIdTag);
    /* vaildity check */
    if (smIndex == -1) {
        fprintf(stderr, "getShaSesmFrIdRespHandler()->can't locate sesMgr!\
            n");
        ShastraIdTagsIn(fd, &inShaIdTags);  /* tags */
        ShastraIdTagsIn(fd, &inShaIdTags);  /* perms */
        return -1;
    }
    pSIdTags = getSesmFrontSIdTags(&inShaIdTag);
    ShastraIdTagsIn(fd, pSIdTags);
    pPermTags = getSesmFrontPermTags(&inShaIdTag);
    ShastraIdTagsIn(fd, pPermTags);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHASESMFRID);
    showInfo(sbOutMsgBuf);
    if (debug) {
        outputIdTags(stderr, pSIdTags);
        outputIdTags(stderr, pPermTags);
    }
}

/*
 * Function
 */
int
setShaSesmFrIdRespHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_SHASESMFRID);
    showInfo(sbOutMsgBuf);
}


/*
 * Function
 */
int
helpRespHandler(fd)
    int             fd;
{
    standardHelpRespHandler(fd);
    /* actually receive help info */
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_HELP);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
quitRespHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_QUIT);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
collInviteJoinRespHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_INVITEJOIN);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
collTellJoinRespHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_TELLJOIN);
```

```
        showInfo(sbOutMsgBuf);
    }

    /*
     * Function
     */
    int
    collTellJnRespHandler(fd)
        int             fd;
    {
        shastraIdTag    sIdTag;
        shastraIdTag    smSIdTag;
        shastraId      *pSId;
        int             outFd;

        ShastraIdTagIn(fd, &smSIdTag);
        ShastraIdTagIn(fd, &sIdTag);

        pSId = getSIdByTagInSIds(&sIdTag, pShastraFrontIds);
        if (pSId == NULL) {
            fprintf(stderr, "collTellJoinHandler()-> no such client!!\n");
            return;
        }
        outFd = shaFrontId2Fd(pSId);
        if (outFd == -1) {
            fprintf(stderr, "collTellJoinHandler()-> no channel for client!!\n"
                );
            return;
        }
        putCollTellJoinHandler(outFd, &smSIdTag, &sIdTag);

        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_TELLJOIN);
        showInfo(sbOutMsgBuf);
    }

    /*
     * Function
     */
    int
    collAskJnRespHandler(fd)
        int             fd;
    {
        shastraIdTag    sIdTag;
        shastraIdTag    smSIdTag;
        shastraIdTag    permsTag;
        shastraId      *pSId;
        int             outFd;

        ShastraIdTagIn(fd, &smSIdTag);
        ShastraIdTagIn(fd, &sIdTag);
        permsTag = 0xff;
        /*
         * pSIdTagHead = &sesMgrStartIdTags.shastraIdTags_val[0]; pSIdTagHead
     * = &pShastraFrontIds->shastraIds_val[0]->lSIDTag;
     */
    /* CHECK actually explicitly store the head honcho */
    if (pShastraFrontIds->shastraIds_len == 0) {
        collTellJoinOprn(&smSIdTag, &sIdTag, &permsTag);
    } else {        /* have someone */
        pSId = pShastraFrontIds->shastraIds_val[0];
        outFd = shaFrontId2Fd(pSId);
        if (outFd == -1) {
            fprintf(stderr, "collAskJnHandler()-> no channel for client!!\
                n");
            return;
        }
        putCollAskJoinHandler(outFd, &smSIdTag, &sIdTag);
    }

    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_ASKJOIN);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int collAskJoinMsgRespHandler(fd)
    int fd;
{
    /* receive sesm idtag, display recvd message */
    shastraIdTag    smSIdTag;
    shastraIdTag    sIdTag;
    shastraIdTag    toSIdTag;
    shastraId      *pSId;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    /*handle*/
    if (pShastraFrontIds->shastraIds_len != 0) {
        pSId = pShastraFrontIds->shastraIds_val[0];
        toSIdTag = pSId->lSIDTag;
    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collAskJoinMsgRespHandler()")){
        case route_FRONT:
            putCollAskJoinMsgHandler(outFd, &smSIdTag, &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    }
    sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COLL_ASKJOINMSG);
    showInfo(sbOutMsgBuf);
```

```
}
/*
 * Function
 */
int collAskJnRespMsgRespHandler(fd)
    int fd;
{
    /* receive sesm idtag, display recvd message */
    shastraIdTag    smSIdTag;
    shastraIdTag    sIdTag;
    shastraIdTag    toSIdTag;
    shastraId      *pSId;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    /*handle*/
    if (pShastraFrontIds->shastraIds_len != 0) {
        pSId = pShastraFrontIds->shastraIds_val[0];
        toSIdTag = pSId->lSIDTag;
    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collAskJnRespMsgRespHandler()")){
        case route_FRONT:
            putCollAskJnRespMsgHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    }
    sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COLL_ASKJNRESPMSG);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int collAskJnStatusRespHandler(fd)
    int fd;
{
    /* receive sesm idtag, display recvd status */
    shastraIdTag    smSIdTag;
    shastraIdTag    sIdTag;
    shastraIdTag    toSIdTag;
    shastraId      *pSId;
    shaULong        lStatus;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    ShastraULongIn(fd, &lStatus);
    /*handle*/
    if (pShastraFrontIds->shastraIds_len != 0) {
        pSId = pShastraFrontIds->shastraIds_val[0];
        toSIdTag = pSId->lSIDTag;
    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collAskJnStatusRespHandler()")){
        case route_FRONT:
            putCollAskJnStatusHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, lStatus);
        break;
        case route_ERROR:
        default:
        break;
    }
    }
    sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COLL_ASKJNSTATUS);
    showInfo(sbOutMsgBuf);
}
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
#include <stdio.h>
#include <sys/errno.h>

#include <shastra/shastra.h>

#include <shastra/utils/hash.h>

#include <shastra/uitools/chooseOne.h>
#include <shastra/uitools/chooseMany.h>
#include <shastra/uitools/callbackArg.h>

#include <shastra/network/server.h>
#include <shastra/network/mplex.h>
#include <shastra/network/hostMgr.h>
#include <shastra/network/sharedMem.h>

#include <shastra/datacomm/shastraDataH.h>
#include <shastra/datacomm/shastraIdH.h>
#include <shastra/datacomm/shastraIdTagH.h>
#include <shastra/datacomm/videoImgH.h>
#include <shastra/datacomm/audioBiteH.h>
#include <shastra/datacomm/pictDataH.h>
#include <shastra/datacomm/xsCntlDataH.h>
#include <shastra/datacomm/ipimage.h>

#include <shastra/shautils/shautils.h>
#include <shastra/shautils/kernelFronts.h>
#include <shastra/shautils/sesMgrFrontsP.h>
#include <shastra/shautils/sesMgrFronts.h>
```

```
#include <shastra/session/sesMgr.h>
#include <shastra/session/sesMgrMainCB.h>
#include <shastra/session/sesMgr_server.h>
#include <shastra/session/sesMgr_client.h>

#define USESHAREDMEM
extern int      debug;
extern collabData *pSesMgrCollData;
extern sesmFronts     *pSesmFrontCD;
collabCommData *pTextCommData;

#define putStringOnChannel(filedesc, reqstr, funcstr)       \
    if (cmSendString(filedesc, reqstr) == -1) {     \
        fprintf(stderr, "%s : Error Sending to %d\n", funcstr, filedesc);
                \
        closedChannelCleanUpHandler(filedesc);                      \
        return;                                         \
    }

#define sendDataString(fd, s)       \
    if(cmSendString(fd, s) == -1){  \
        fprintf(stderr,"Error in Sending Operation Data\n");    \
        closedChannelCleanUpHandler(fd);        \
        return;                                                         \
    }


#define ShastraIdIn(filedesc, pShaId)           \
    if(shastraIdIn(filedesc, pShaId) == -1){    \
        fprintf(stderr, "Error Receiving SID from %d\n", filedesc); \
        closedChannelCleanUpHandler(filedesc);                  \
        return;                                         \
    }

#define ShastraIdOut(filedesc, pShaId)          \
    if(shastraIdOut(filedesc, pShaId) == -1){   \
        fprintf(stderr, "Error Sending SID to %d\n", filedesc); \
        closedChannelCleanUpHandler(filedesc);                  \
        return;                                         \
    }

#define ShastraIdsIn(filedesc, pShaIds)         \
    if(shastraIdsIn(filedesc, pShaIds) == -1){  \
        fprintf(stderr, "Error Receiving SIDs from %d\n", filedesc);    \
        closedChannelCleanUpHandler(filedesc);                  \
        return;                                         \
    }

#define ShastraIdsOut(filedesc, pShaIds)            \
    if(shastraIdsOut(filedesc, pShaIds) == -1){ \
        fprintf(stderr, "Error Sending SIDs to %d\n", filedesc);    \
        closedChannelCleanUpHandler(filedesc);                  \
            return;                                          \
        }


    #define ShastraIdTagIn(filedesc, pShaIdTag)         \
        if(shastraIdTagIn(filedesc, pShaIdTag) == -1){  \
            fprintf(stderr, "Error Receiving SID from %d\n", filedesc); \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                          \
        }

    #define ShastraIdTagOut(filedesc, pShaIdTag)             \
        if(shastraIdTagOut(filedesc, pShaIdTag) == -1){ \
            fprintf(stderr, "Error Sending SID to %d\n", filedesc); \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                          \
        }

    #define ShastraIdTagsIn(filedesc, pShaIdTags)            \
        if(shastraIdTagsIn(filedesc, pShaIdTags) == -1){     \
            fprintf(stderr, "Error Receiving SIDs from %d\n", filedesc);    \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                          \
        }

    #define ShastraIdTagsOut(filedesc, pShaIdTags)           \
        if(shastraIdTagsOut(filedesc, pShaIdTags) == -1){    \
            fprintf(stderr, "Error Sending SIDs to %d\n", filedesc);    \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                          \
        }

    #define VideoImgIn(filedesc, pVImg)         \
        if(videoImgIn(filedesc, pVImg) == -1){  \
            fprintf(stderr, "Error Receiving VImg from %d\n", filedesc);    \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                          \
        }

    #define VideoImgOut(filedesc, pVImg)            \
        if(videoImgOut(filedesc, pVImg) == -1){ \
            fprintf(stderr, "Error Sending VImg to %d\n", filedesc);    \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                          \
        }

    #define AudioBiteIn(filedesc, pABite)           \
        if(audioBiteIn(filedesc, pABite) == -1){    \
            fprintf(stderr, "Error Receiving ABite from %d\n", filedesc);   \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                          \
        }
    #define AudioBiteOut(filedesc, pABite)          \
        if(audioBiteOut(filedesc, pABite) == -1){   \
            fprintf(stderr, "Error Sending ABite to %d\n", filedesc);   \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                         \
        }

    #define ImageDataIn(filedesc, pImage)            \
        if(ipimageDataIn(filedesc, pImage) == -1){  \
            fprintf(stderr, "Error Receiving image from %d\n", filedesc); \
            closedChannelCleanUpHandler(filedesc);       \
            return;                                         \
        }

    #define ImageDataOut(filedesc, pImage)           \
        if(ipimageDataOut(filedesc, pImage) == -1){ \
            fprintf(stderr, "Error Sending image to %d\n", filedesc); \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                         \
        }


    #define ShastraULongOut(filedesc, pULong)           \
        if(shaULongOut(filedesc, pULong) == -1){ \
            fprintf(stderr, "Error Sending pULong to %d\n", filedesc);    \
            closedChannelCleanUpHandler(filedesc);   \
            return;                                         \
        }

    #define ShastraULongIn(filedesc, pULong)            \
        if(shaULongIn(filedesc, pULong) == -1){ \
            fprintf(stderr, "Error Receiving pULong from %d\n", filedesc);    \
            closedChannelCleanUpHandler(filedesc);   \
            return;                                         \
        }

    #define ShastraIntOut(filedesc, pInt)           \
        if(shaIntOut(filedesc, pInt) == -1){ \
            fprintf(stderr, "Error Sending pInt to %d\n", filedesc);    \
            closedChannelCleanUpHandler(filedesc);   \
            return;                                         \
        }

    #define ShastraIntIn(filedesc, pInt)           \
        if(shaIntIn(filedesc, pInt) == -1){ \
            fprintf(stderr, "Error Receiving pInt from %d\n", filedesc);    \
            closedChannelCleanUpHandler(filedesc);   \
            return;                                         \
        }

    #define PictDataBitesIn(filedesc, pPCDatas)         \
        if(pictPiecesIn(filedesc, pPCDatas) == -1){ \
            fprintf(stderr, "Error Receiving PCDatas from %d\n", filedesc); \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                         \
        }

    #define PictDataBitesOut(filedesc, pPCDatas)            \
        if(pictPiecesOut(filedesc, pPCDatas) == -1){    \
            fprintf(stderr, "Error Sending PCDatas to %d\n", filedesc); \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                         \
        }

    #define XSCntlBitesIn(filedesc, pXSCDatas)          \
        if(xsCntlDatasIn(filedesc, pXSCDatas) == -1){   \
            fprintf(stderr, "Error Receiving XSCDatas from %d\n", filedesc);
                    \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                         \
        }

    #define XSCntlBitesOut(filedesc, pXSCDatas)         \
        if(xsCntlDatasOut(filedesc, pXSCDatas) == -1){  \
            fprintf(stderr, "Error Sending XSCDatas to %d\n", filedesc);    \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                         \
        }

    #define PntrBiteIn(filedesc, pABite)            \
        if(shaDoublesIn(filedesc, pABite) == -1){   \
            fprintf(stderr, "Error Receiving PntrB from %d\n", filedesc);   \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                         \
        }

    #define PntrBiteOut(filedesc, pABite)           \
        if(shaDoublesOut(filedesc, pABite) == -1){  \
            fprintf(stderr, "Error Sending PntrB to %d\n", filedesc);   \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                         \
        }

    #define CursorBiteIn(filedesc, pABite)          \
        if(shaDoublesIn(filedesc, pABite) == -1){   \
            fprintf(stderr, "Error Receiving CursorB from %d\n", filedesc); \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                         \
        }

    #define CursorBiteOut(filedesc, pABite)         \
        if(shaDoublesOut(filedesc, pABite) == -1){  \
            fprintf(stderr, "Error Sending CursorB to %d\n", filedesc); \
            closedChannelCleanUpHandler(filedesc);                  \
            return;                                         \
        }
```

```
shaRouteMode
routeFrontSIdTagToFd(pSIdTag, pFd, nmFunc)
    shastraIdTag *pSIdTag;
    int *pFd;
    char *nmFunc;
{
    shastraId *pSId;
    int outFd = -1;
    shaRouteMode retVal = route_ERROR;

    pSId = getSIdByTagInSIds(pSIdTag, pShastraFrontIds);
    if (pSId == NULL) {
        sprintf(sbOutMsgBuf, "%s->Unknown IDTag -- Aborted\n", nmFunc);
        showInfo(sbOutMsgBuf);
        return retVal;
    }
    outFd = shaFrontId2Fd(pSId);
    if (outFd == -1) {
        sprintf(sbOutMsgBuf, "%s->Unknown Front -- Aborted\n", nmFunc);
        showInfo(sbOutMsgBuf);
        return retVal;
    }
    else{
        retVal = route_FRONT;
    }
    *pFd = outFd;
    return retVal;
}


helpHandler(fd)
    int             fd;
{
    int             i;
    char            buf[512];

    cmAckOk(fd);
    sprintf(buf, "%d\n", serverNCmds);
    putStringOnChannel(fd, buf, "helpHandler()");
    for (i = 0; i < serverNCmds; i++) {
        sprintf(buf, "%s -- %s\n", serverCommandTab[i].command,
            serverCommandTab[i].helpmsg);
        putStringOnChannel(fd, buf, "helpHandler()");
    }
    cmFlush(fd);

    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_HELP);
    showInfo(sbOutMsgBuf);
}
```

```
terminateHandler(fd)
    int             fd;
{
    char           *buf;

    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_TERMINATE);
    showInfo(sbOutMsgBuf);
    quitOprn(0);
}

collTerminateHandler(fd)
    int             fd;
{
    int i;

    cmAckOk(fd);
    cmFlush(fd);

    {
        int            *pfd;
        int             nfd;

        getKrFDsBCast(&pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollLeaveHandler, NULL);
        for(i=0;i<nfd;i++){
            localShaIdIn[pfd[i]].lSIDTag = 0;
        }
    }
    sleep(2);
    quitOprn(0);
    return 0;

    updateShaFrontIds(pShastraFrontIds);
    krFrSIds2SIdTags(pShastraFrontIds, pShastraFrontIdTags);
    krFrSIds2PermTags(pShastraFrontIds, pShastraFrontPermTags);

    if (rgsbShastraFront != NULL) {
        strListDestroy(rgsbShastraFront);
    }
    rgsbShastraFront = pSIds2StrTab(pShastraFrontIds, PSIDNMHOST |
        PSIDNMAPPL);
    chooseOneChangeList(pcoShastraFront, rgsbShastraFront,
                coNoInitialHighlight);

    if (collabTerminateFunc != NULL) {
        (*collabTerminateFunc) ();
    }
    setShaSesmFrIdOprn(0);
    sleep(2);
    quitOprn(0);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_TERMINATE);
    showInfo(sbOutMsgBuf);
    }

    collRemoveHandler(fd)
        int             fd;
    {
        int             outFd;
        shastraId      *pSId;
        shastraIdTag    sIdTag;

        ShastraIdTagIn(fd, &sIdTag);
        cmAckOk(fd);
        cmFlush(fd);

        pSId = getSIdByTagInSIds(&sIdTag, pShastraFrontIds);
        if (pSId == NULL) {
            fprintf(stderr, "collRemoveHandler()-> no such client!!\n");
            return;
        }
        outFd = shaFrontId2Fd(pSId);
        if (outFd == -1) {
            fprintf(stderr, "collRemoveHandler()-> no channel for client!!\n");
            return;
        }
        putCollLeaveHandler(outFd);

        collLeaveCleanUpHandler(outFd);
        shaKernFlags[outFd] = 0;
        localShaIdIn[outFd].lSIDTag = 0;
        updateShaFrontIds(pShastraFrontIds);

        if (collabRemoveFunc != NULL) {
            (*collabRemoveFunc) ();
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_REMOVE);
        showInfo(sbOutMsgBuf);
    }

    collTellJoinHandler(fd)
        int             fd;
    {
        shastraIdTag    sIdTag;
        shastraIdTag    smSIdTag;
        shastraIdTag    permsTag;
        shastraId      *pSId;
        int             outFd;

        ShastraIdTagIn(fd, &smSIdTag);
        ShastraIdTagIn(fd, &sIdTag);
        ShastraIdTagIn(fd, &permsTag);

        cmAckOk(fd);
        cmFlush(fd);
        collTellJoinOprn(&smSIdTag, &sIdTag, &permsTag);

        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_TELLJOIN);
        showInfo(sbOutMsgBuf);
    }

    collJoinHandler(fd)
        int             fd;
    {
        shastraId       *pSId;
        extern shastraIdTags *pShastraFrontIdTags;
        extern unsigned long kernelIdTag;
        collabFrontData *pCollFrData;

        pSId = &localShaIdIn[fd];
        shaKernFlags[fd] = SHAFRONT;
        ShastraIdIn(fd, pSId);
        if (debug) {
            outputId(stderr, pSId);
        }

        updateShaFrontIds(pShastraFrontIds);
        krFrSIds2SIdTags(pShastraFrontIds, pShastraFrontIdTags);
        krFrSIds2PermTags(pShastraFrontIds, pShastraFrontPermTags);

        if (occupySmFrFrontFreeSlot( & kernelShastraId.lSIDTag,
                & pSId->lSIDTag) < 0) {
            fprintf(stderr, "collJoinHandler()->couldn't
                occupySmFrFrontFreeSlot!\n");
        }
        pCollFrData = (collabFrontData *) malloc(sizeof(collabFrontData));
        memset(pCollFrData, 0, sizeof(collabFrontData));
        if (getSesMgrFrontData(
                        & kernelShastraId.lSIDTag,
                        & pSId->lSIDTag) != NULL) {
            fprintf(stderr, "collJoinHandler()->warning.. has SesMgrFrontData!\
                n");
        }
        if (setSesMgrFrontData( & kernelShastraId.lSIDTag,
              & pSId->lSIDTag, (char *) pCollFrData) < 0) {
            fprintf(stderr, "collJoinHandler()->couldn't setSesMgrFrontData!\n"
                );
        }
        if (rgsbShastraFront != NULL) {
            strListDestroy(rgsbShastraFront);
        }
        rgsbShastraFront = pSIds2StrTab(pShastraFrontIds, PSIDNMHOST |
            PSIDNMAPPL);
        chooseOneChangeList(pcoShastraFront, rgsbShastraFront,
                    coNoInitialHighlight);

        setShaSesmFrIdOprn(0);
        sleep(1);
```

```
/*
    if(pSId->lSIDTag == sesMgrStartIdTags.shastraIdTags_val[0])
*/
    if(pSId->lSIDTag == pShastraFrontIdTags->shastraIdTags_val[0])
    {
        putCollTellLeaderHandler(fd, &kernelShastraId.lSIDTag,
            &pSId->lSIDTag, &kernelIdTag);
    }
    cmAckOk(fd);
    cmFlush(fd);
#ifdef WANTTHIS
    putShaSesmFrIdHandler(fd, & kernelShastraId.lSIDTag);
    {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putShaSesmFrIdHandler,
                (char *) &kernelShastraId.lSIDTag);
    }
#endif              /* WANTTHIS */
    if (collabJoinFunc != NULL) {
        (*collabJoinFunc) ();
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_JOIN);
    showInfo(sbOutMsgBuf);
}

collLeaveHandler(fd)
    int             fd;
{
    collLeaveCleanUpHandler(fd);
}

collLeaveCleanUpHandler(fd)
    int             fd;
{
    int             fKern;
    extern shastraIdTags *pShastraFrontIdTags;
    shastraId      *pSId;
    collabFrontData *pCollFrData;

    pSId = &localShaIdIn[fd];
    shMemDisconnect(mplexInShmInfo(fd));
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
                & kernelShastraId.lSIDTag,
                & pSId->lSIDTag);
    if (pCollFrData != NULL) {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        if (pCollFrData->fTextState == COMM_STARTED) {
                cmMultiCast(pfd, nfd, putCollEndTextHandler,
                        (char *) &localShaIdIn[fd].lSIDTag);
            }
            if (pCollFrData->fAudioState == COMM_STARTED) {
                cmMultiCast(pfd, nfd, putCollEndAudioHandler,
                        (char *) &localShaIdIn[fd].lSIDTag);
            }
            if (pCollFrData->fVideoState == COMM_STARTED) {
                cmMultiCast(pfd, nfd, putCollEndVideoHandler,
                        (char *) &localShaIdIn[fd].lSIDTag);
            }
            if (pCollFrData->fPolyState == COMM_STARTED) {
                cmMultiCast(pfd, nfd, putCollEndPolyHandler,
                        (char *) &localShaIdIn[fd].lSIDTag);
            }
            if (pCollFrData->fXSCntlState == COMM_STARTED) {
                cmMultiCast(pfd, nfd, putCollEndXSCntlHandler,
                        (char *) &localShaIdIn[fd].lSIDTag);
            }
            if (pCollFrData->fPntrState == COMM_STARTED) {
                cmMultiCast(pfd, nfd, putCollEndPntrHandler,
                        (char *) &localShaIdIn[fd].lSIDTag);
            }
            if (pCollFrData->fCursorState == COMM_STARTED) {
                cmMultiCast(pfd, nfd, putCollEndCursorHandler,
                        (char *) &localShaIdIn[fd].lSIDTag);
            }
            if (pCollFrData->fPictState == COMM_STARTED) {
                cmMultiCast(pfd, nfd, putCollEndPictHandler,
                        (char *) &localShaIdIn[fd].lSIDTag);
            }
            memset(pCollFrData, 0, sizeof(collabFrontData));
            free(pCollFrData);
        }
        if (setSesMgrFrontData( & kernelShastraId.lSIDTag,
                 & pSId->lSIDTag, (char *) NULL) < 0) {
            fprintf(stderr, "collJoinHandler()->couldn't setSesMgrFrontData!\n"
                );
        }
        if (freeSmFrFrontSlot( & kernelShastraId.lSIDTag,
                      & pSId->lSIDTag) < 0) {
            fprintf(stderr, "collJoinHandler()->couldn't freeSmFrFrontSlot!\n")
                ;
        }
        fKern = shaKernFlags[fd];
        deleteShaIdFromTab(fd, pShastraFrontIds);
        mplexUnRegisterChannel(fd);

        krFrSIds2SIdTags(pShastraFrontIds, pShastraFrontIdTags);
        krFrSIds2PermTags(pShastraFrontIds, pShastraFrontPermTags);

        if (fKern != SHAFRONT) {
            fprintf(stderr, "collLeaveCleanUpHandler()-> shouldn't happen!\n");
        return;
    } else {
        if (rgsbShastraFront != NULL) {
            strListDestroy(rgsbShastraFront);
        }
        rgsbShastraFront = pSIds2StrTab(pShastraFrontIds,
                        PSIDNMHOST | PSIDNMAPPL);
        chooseOneChangeList(pcoShastraFront, rgsbShastraFront,
                    coNoInitialHighlight);

        setShaSesmFrIdOprn(0);
#ifdef WANTTHIS
        {
            int             *pfd;
            int              nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putShaSesmFrIdHandler,
                    (char *) &kernelShastraId.lSIDTag);
        }
#endif              /* WANTTHIS */

    }
/* CHECK --alos, go into comm record and cause buffer release */
    if (pTextCommData != NULL) {
        if (pTextCommData->nMembers > 0) {
            pTextCommData->nMembers--;
        }
    }
    if (collabLeaveFunc != NULL) {
        (*collabLeaveFunc) ();
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_LEAVE);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */

int
oldcollStartTextHandler(fd)
    int             fd;
{
    cmAckOk(fd);
    cmFlush(fd);

    if (pTextCommData != NULL) {
        return;
    }
    pTextCommData = (collabCommData *) malloc(sizeof(collabCommData));
    memset(pTextCommData, 0, sizeof(collabCommData));
    pTextCommData->nMembers = pShastraFrontIdTags->shastraIdTags_len;
    pTextCommData->htCommBufs = htMakeNew(COMMHASHTABLESIZE, 0 );

    {
        int             *pfd;
        int              nfd;
        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollStartTextHandler,
                (char *) NULL);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_START_TEXT);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
oldcollEndTextHandler(fd)
    int             fd;
{
    cmAckOk(fd);
    cmFlush(fd);

    if (pTextCommData == NULL) {
        return;
    }
    htDestroy(pTextCommData->htCommBufs, 1);
    free(pTextCommData);
    pTextCommData = NULL;

    {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollEndTextHandler,
                (char *) NULL);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_END_TEXT);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
oldcollSendTextHandler(fd)
    int             fd;
{
    char           *bufNam;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);
    if (pTextCommData == NULL) {
        return;
    } {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollSendTextHandler,
                bufNam);
    }
    free(bufNam);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_TEXT);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
oldcollSendMsgTextHandler(fd)
    int             fd;
{
    char            *bufNam;
    collabCommRecordData *pCommRec;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    if (pTextCommData == NULL) {
    } else {
        pCommRec = (collabCommRecordData *) malloc(sizeof
            (collabCommRecordData));
        memset(pCommRec, 0, sizeof(collabCommRecordData));
        pCommRec->refCount = pTextCommData->nMembers - 1;
        pCommRec->inChannel = fd;
        htInstallSymbol(pTextCommData->htCommBufs, bufNam, (char *)
            pCommRec);
    }
    {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        cmMultiCast(pfd, nfd, putCollSendMsgTextHandler,
                bufNam);
        pSesMgrCollData->pShmInfoOut->shmDirty = 0;
    }
    free(bufNam);
    return;
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGTEXT);
    showInfo(sbOutMsgBuf);
```

```
}
/*
 * Function
 */
int
oldcollRecvdMsgTextHandler(fd)
    int             fd;
{
    char          *bufNam;
    struct he     *phe;
    collabCommRecordData *pCommRec;

    bufNam = cmReceiveString(fd);
    if (pTextCommData == NULL) {
        cmAckOk(fd);
        cmFlush(fd);
        return;
    }
    phe = htLookup(pTextCommData->htCommBufs, bufNam);
    if (phe == NULL) {
        fprintf(stderr, "collRecvdTextHandler()->no such buffer known!\n");
        cmAckError(fd);
        cmFlush(fd);
        return;
    }
    cmAckOk(fd);
    cmFlush(fd);

    pCommRec = (collabCommRecordData *) phe->data;
    pCommRec->refCount--;
    if (pCommRec->refCount <= 0) {
        /* free, free at last */
        putCollRecvdMsgTextHandler(pCommRec->inChannel, bufNam);
        heDelete(pTextCommData->htCommBufs, bufNam);
        free(pCommRec);
        free(bufNam);
    }
    return;
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGTEXT);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */

int
collStartTextHandler(fd)
    int             fd;
{
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    cmAckOk(fd);
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);

    pSesMgrCollData->fTextState = COMM_STARTED;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if (pCollFrData != NULL) {
        pCollFrData->fTextState = COMM_STARTED;
    } else {
        fprintf(stderr, "collStartTextHandler()->no SmFrData!");
    }
    {
        int            *pfd;
        int             nfd;
        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollStartTextHandler,
                (char *) &localShaIdIn[fd].lSIDTag);
    }

    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_START_TEXT);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collEndTextHandler(fd)
    int             fd;
{
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    cmAckOk(fd);
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);

    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if (pCollFrData != NULL) {
        pCollFrData->fTextState = COMM_ENDED;
    } else {
        fprintf(stderr, "collStartTextHandler()->no SmFrData!");
    }
    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollEndTextHandler,
                (char *) &localShaIdIn[fd].lSIDTag);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_END_TEXT);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendTextHandler(fd)
    int             fd;
{
    char           *bufNam;
    bunchOfThings   bunch;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    bunch.nThings = 2;
    bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
    bunch.things[1] = bufNam;
    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollSendTextHandler,
                (char *) &bunch);
    }
    free(bufNam);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_TEXT);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendMsgTextHandler(fd)
    int             fd;
{
    bunchOfThings   bunch;
    char           *buf;
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    buf = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fTextState == COMM_ENDED)) {
    } else {
        bunch.nThings = 2;
        bunch.things[0] = (char *) pSIdTag;
        bunch.things[1] = buf;
        {
            int            *pfd;
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgTextHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
    free(buf);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGTEXT);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collRecvdMsgTextHandler(fd)
    int             fd;
{
    char           *bufNam;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    free(bufNam);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGTEXT);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendMsgShmTextHandler(fd)
    int             fd;
{
    int             shmId;
    bunchOfThings   bunch;
    char           *buf;
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;
    shmInfo        *pShmInfo;
    int             n;

    ShastraIntIn(fd, &shmId);
    cmAckOk(fd);
    cmFlush(fd);
    if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
        fprintf(stderr, "collSendMsgShmTextHandler()->no non-local SHM\n");
        return;
    }
    pShmInfo = mplexInShmInfo(fd);
    if (!shMemReconnect(pShmInfo, shmId)) {
        fprintf(stderr, "collSendMsgShmTextHandler()->SHM recon problem\n")
            ;
        return;
    }
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fTextState == COMM_ENDED)) {
    } else {
        buf = pShmInfo->shmAddr;
        bunch.nThings = 2;
        bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
        bunch.things[1] = buf;
        {
            int            *pfd;
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgTextHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMTEXT);
    showInfo(sbOutMsgBuf);

}
/*
 * Function
 */
int
collRecvdMsgShmTextHandler(fd)
    int             fd;
{
    int             shmId;

    ShastraIntIn(fd, &shmId);
    cmAckOk(fd);
    cmFlush(fd);

    if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
        fprintf(stderr, "collRecvdMsgShmTextHandler()->no non-local SHM\n")
            ;
        return;
    }
```

```
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMTEXT);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */

    int
    collStartAudioHandler(fd)
        int             fd;
    {
        shastraIdTag   *pSIdTag;
        collabFrontData *pCollFrData;

        cmAckOk(fd);
        pSIdTag = & localShaIdIn[fd].lSIDTag;
        ShastraIdTagOut(fd, pSIdTag);
        cmFlush(fd);

        pSesMgrCollData->fAudioState = COMM_STARTED;
        pCollFrData = (collabFrontData *) getSesMgrFrontData(
                    & kernelShastraId.lSIDTag, pSIdTag);
        if (pCollFrData != NULL) {
            pCollFrData->fAudioState = COMM_STARTED;
        } else {
            fprintf(stderr, "collStartAudioHandler()->no SmFrData!");
        }
        {
            int             *pfd;
            int              nfd;
            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putCollStartAudioHandler,
                    (char *) &localShaIdIn[fd].lSIDTag);
        }

        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_START_AUDIO);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */
    int
    collEndAudioHandler(fd)
        int             fd;
    {
        shastraIdTag   *pSIdTag;
        collabFrontData *pCollFrData;

        cmAckOk(fd);
        pSIdTag = & localShaIdIn[fd].lSIDTag;
        ShastraIdTagOut(fd, pSIdTag);
        cmFlush(fd);
        pCollFrData = (collabFrontData *) getSesMgrFrontData(
                    & kernelShastraId.lSIDTag, pSIdTag);
        if (pCollFrData != NULL) {
            pCollFrData->fAudioState = COMM_ENDED;
        } else {
            fprintf(stderr, "collStartAudioHandler()->no SmFrData!");
        }
        {
            int             *pfd;
            int              nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putCollEndAudioHandler,
                    (char *) &localShaIdIn[fd].lSIDTag);
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_END_AUDIO);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */
    int
    collSendAudioHandler(fd)
        int             fd;
    {
        char            *bufNam;
        bunchOfThings   bunch;

        bufNam = cmReceiveString(fd);
        cmAckOk(fd);
        cmFlush(fd);

        bunch.nThings = 2;
        bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
        bunch.things[1] = bufNam;
        {
            int             *pfd;
            int              nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putCollSendAudioHandler,
                    (char *) &bunch);
        }
        free(bufNam);
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_AUDIO);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */
    int
    collSendMsgAudioHandler(fd)
        int             fd;
```

```
{
    bunchOfThings   bunch;
    char           *buf;
    static audioBite aBite;
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    AudioBiteIn(fd, &aBite);
    cmAckOk(fd);
    cmFlush(fd);

    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fAudioState == COMM_ENDED))
        {
    } else {
        bunch.nThings = 2;
        bunch.things[0] = (char *) pSIdTag;
        bunch.things[1] = (char *) &aBite ;
        {
            int            *pfd;
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgAudioHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
    return;
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGAUDIO);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collRecvdMsgAudioHandler(fd)
    int             fd;
{
    char           *bufNam;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    free(bufNam);
    return;
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGAUDIO);
    showInfo(sbOutMsgBuf);
}
```

```
/*
 * Function
 */
int
collSendMsgShmAudioHandler(fd)
    int             fd;
{
    int             shmId;
    bunchOfThings   bunch;
    static audioBite aBite;
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;
    shmInfo        *pShmInfo;
    int             n;

    ShastraIntIn(fd, &shmId);
    cmAckOk(fd);
    cmFlush(fd);

    if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
        fprintf(stderr, "collSendMsgShmAudioHandler()->no non-local SHM\n")
            ;
        return;
    }
    pShmInfo = mplexInShmInfo(fd);
    if (!shMemReconnect(pShmInfo, shmId)) {
        fprintf(stderr, "collSendMsgShmAudioHandler()->SHM recon problem\n"
            );
        return;
    }
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fAudioState == COMM_ENDED))
        {
    } else {

        audioBiteMemIn(pShmInfo->shmAddr, pShmInfo->shmSize,
                   &aBite);
        bunch.nThings = 2;
        bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
        bunch.things[1] = (char *) &aBite;
        {
            int             *pfd;
            int              nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgAudioHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
```

```
        return;
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMAUDIO);
        showInfo(sbOutMsgBuf);

    }
    /*
     * Function
     */
    int
    collRecvdMsgShmAudioHandler(fd)
        int             fd;
    {
        int             shmId;

        ShastraIntIn(fd, &shmId);
        cmAckOk(fd);
        cmFlush(fd);

        if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
            fprintf(stderr, "collRecvdMsgShmAudioHandler()->no non-local SHM\n"
                );
            return;
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMAUDIO);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */

    int
    collStartVideoHandler(fd)
        int             fd;
    {
        shastraIdTag   *pSIdTag;
        collabFrontData *pCollFrData;

        cmAckOk(fd);
        pSIdTag = & localShaIdIn[fd].lSIDTag;
        ShastraIdTagOut(fd, pSIdTag);
        cmFlush(fd);

        pSesMgrCollData->fVideoState = COMM_STARTED;
        pCollFrData = (collabFrontData *) getSesMgrFrontData(
                    & kernelShastraId.lSIDTag, pSIdTag);
        if (pCollFrData != NULL) {
            pCollFrData->fVideoState = COMM_STARTED;
        } else {
            fprintf(stderr, "collStartVideoHandler()->no SmFrData!");
        }
        {
            int             *pfd;
            int              nfd;
        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollStartVideoHandler,
                (char *) &localShaIdIn[fd].lSIDTag);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_START_VIDEO);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collEndVideoHandler(fd)
    int             fd;
{
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    cmAckOk(fd);
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);

    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if (pCollFrData != NULL) {
        pCollFrData->fVideoState = COMM_ENDED;
    } else {
        fprintf(stderr, "collStartVideoHandler()->no SmFrData!");
    }
    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollEndVideoHandler,
                (char *) &localShaIdIn[fd].lSIDTag);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_END_VIDEO);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendVideoHandler(fd)
    int             fd;
{
    char           *bufNam;
    bunchOfThings   bunch;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);
    bunch.nThings = 2;
    bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
    bunch.things[1] = bufNam;
    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollSendVideoHandler,
                (char *) &bunch);
    }
    free(bufNam);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_VIDEO);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendMsgVideoHandler(fd)
    int             fd;
{
    bunchOfThings   bunch;
    char           *bufNam;
    static videoImg vImg;
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    VideoImgIn(fd, &vImg);
    cmAckOk(fd);
    cmFlush(fd);

    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fVideoState == COMM_ENDED))
        {
    } else {
        bunch.nThings = 2;
        bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
        bunch.things[1] = (char *) &vImg;
        {
            int            *pfd;
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgVideoHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
```

```
        return;
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGVIDEO);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */
    int
    collRecvdMsgVideoHandler(fd)
        int             fd;
    {
        char           *bufNam;

        bufNam = cmReceiveString(fd);
        cmAckOk(fd);
        cmFlush(fd);

        free(bufNam);
        return;
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGVIDEO);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */
    int
    collSendMsgShmVideoHandler(fd)
        int             fd;
    {
        int             shmId;
        bunchOfThings   bunch;
        static videoImg vImg;
        shastraIdTag   *pSIdTag;
        collabFrontData *pCollFrData;
        shmInfo        *pShmInfo;
        int             n;

        ShastraIntIn(fd, &shmId);
        cmAckOk(fd);
        cmFlush(fd);

        if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
            fprintf(stderr, "collSendMsgShmVideoHandler()->no non-local SHM\n")
                ;
            return;
        }
        pShmInfo = mplexInShmInfo(fd);
        if (!shMemReconnect(pShmInfo, shmId)) {
            fprintf(stderr, "collSendMsgShmVideoHandler()->SHM recon problem\n"
                );
            return;
        }
        pSIdTag = & localShaIdIn[fd].lSIDTag;
        pCollFrData = (collabFrontData *) getSesMgrFrontData(
                    & kernelShastraId.lSIDTag, pSIdTag);
        if ((pCollFrData == NULL) || (pCollFrData->fVideoState == COMM_ENDED))
            {
        } else {

            videoImgMemIn(pShmInfo->shmAddr, pShmInfo->shmSize,
                    &vImg);
            bunch.nThings = 2;
            bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
            bunch.things[1] = (char *) &vImg;
            {
                int             *pfd;
                int              nfd;

                getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
                pSesMgrCollData->pShmInfoOut->shmDirty = 0;
                cmMultiCast(pfd, nfd, putCollSendMsgVideoHandler,
                        (char *) &bunch);
                pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            }
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMVIDEO);
        showInfo(sbOutMsgBuf);

    }
    /*
     * Function
     */
    int
    collRecvdMsgShmVideoHandler(fd)
        int             fd;
    {
        int             shmId;

        ShastraIntIn(fd, &shmId);
        cmAckOk(fd);
        cmFlush(fd);

        if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
            fprintf(stderr, "collRecvdMsgShmVideoHandler()->no non-local SHM\n"
                );
            return;
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMVIDEO);
        showInfo(sbOutMsgBuf);
    }

    /*
     * Function
     */
    int
    collGetPermsHandler(fd)
    int             fd;
{
    shastraIdTag    sIdTag;
    int             iFr;

    ShastraIdTagIn(fd, &sIdTag);

    iFr = getSIdTagIndexInSIdTags(&sIdTag, pShastraFrontIdTags);
    if (iFr == -1) {
        fprintf(stderr, "collGetPermsHandler()->no such front %lx\n",
            sIdTag);
        cmAckError(fd);
        cmFlush(fd);
    } else {
        cmAckOk(fd);
        ShastraIdTagOut(fd, &sIdTag);
        ShastraIdTagOut(fd, &pShastraFrontPermTags->shastraIdTags_val[iFr])
            ;
        cmFlush(fd);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_COLLPERMS);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
collSetPermsHandler(fd)
    int             fd;
{
    shastraIdTag    sIdTag;
    shastraIdTag    permTag;
    int             iFr;

    ShastraIdTagIn(fd, &sIdTag);
    ShastraIdTagIn(fd, &permTag);
    iFr = getSIdTagIndexInSIdTags(&sIdTag, pShastraFrontIdTags);
    if(iFr == 0){
        permTag |= SHASTRA_PERM_GRANT;
    }
    if (iFr == -1) {
        fprintf(stderr, "collSetPermsHandler()->no such front %lx\n",
            sIdTag);
        cmAckError(fd);
        cmFlush(fd);
    } else {
        cmAckOk(fd);
        ShastraIdTagOut(fd, &sIdTag);
        ShastraIdTagOut(fd, &permTag);
        cmFlush(fd);

        pShastraFrontIds->shastraIds_val[iFr]->lPerms = permTag;
        pShastraFrontPermTags->shastraIdTags_val[iFr] = permTag;
        {
            int            *pfd;
            int             nfd;
            bunchOfThings   bunch;

            bunch.nThings = 2;
            bunch.things[0] = (char *) &sIdTag;
            bunch.things[1] = (char *) &permTag;
            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putSetCollPermsHandler,
                    (char *) &bunch);
        }
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_COLLPERMS);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collGetSesmPermsHandler(fd)
    int             fd;
{
    cmAckOk(fd);
    ShastraIdTagOut(fd, & kernelShastraId.lSIDTag);
    ShastraIdTagsOut(fd, pShastraFrontPermTags);
    cmFlush(fd);

    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SESMCOLLPERMS);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
collSetSesmPermsHandler(fd)
    int             fd;
{
    static shastraIdTags permTags;
    shastraIdTag   *pSIdTag;
    int             i;

    ShastraIdTagsIn(fd, &permTags);
    cmAckOk(fd);
    cmFlush(fd);

    if ((pShastraFrontPermTags->shastraIdTags_len ==
         permTags.shastraIdTags_len) &&
        permTags.shastraIdTags_len == pShastraFrontIds->shastraIds_len) {
        for (i = 0; i < pShastraFrontIds->shastraIds_len; i++) {
            pShastraFrontIds->shastraIds_val[i]->lPerms =
                    permTags.shastraIdTags_val[i];
            }
            pSIdTag = pShastraFrontPermTags->shastraIdTags_val;
            pShastraFrontPermTags->shastraIdTags_val = permTags.
                shastraIdTags_val;
            permTags.shastraIdTags_val = pSIdTag;
        } {
            int             *pfd;
            int              nfd;
            bunchOfThings    bunch;

            bunch.nThings = 2;
            bunch.things[0] = (char *) &kernelShastraId.lSIDTag;
            bunch.things[1] = (char *) pShastraFrontPermTags;
            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putSetSesmCollPermsHandler,
                    (char *) &bunch);
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_SESMCOLLPERMS);
        showInfo(sbOutMsgBuf);
    }


    /*
     * Function
     */
    int
    collGetIxnModeHandler(fd)
        int             fd;
    {
        cmAckOk(fd);
        ShastraULongOut(fd, &pSesmFrontCD->lIxnMode);
        cmFlush(fd);

        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_IXNMODE);
        showInfo(sbOutMsgBuf);
    }

    /*
     * Function
     */
    int
    collSetIxnModeHandler(fd)
        int             fd;
    {
        ShastraULongIn(fd, &pSesmFrontCD->lIxnMode);
        cmAckOk(fd);
        ShastraULongOut(fd, &pSesmFrontCD->lIxnMode);
        cmFlush(fd);
        {
            int             *pfd;
            int              nfd;
        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollSetIxnModeHandler,
                (char *) &pSesmFrontCD->lIxnMode);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_IXNMODE);
    showInfo(sbOutMsgBuf);
}


/*
 * Function
 */
int
collGetFloorModeHandler(fd)
    int             fd;
{
    cmAckOk(fd);
    ShastraULongOut(fd, &pSesmFrontCD->lFloorMode);
    cmFlush(fd);

    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_FLOORMODE);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
collSetFloorModeHandler(fd)
    int             fd;
{
    ShastraULongIn(fd, &pSesmFrontCD->lFloorMode);
    cmAckOk(fd);
    ShastraULongOut(fd, &pSesmFrontCD->lFloorMode);
    cmFlush(fd);

    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollSetFloorModeHandler,
                (char *) &pSesmFrontCD->lFloorMode);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_FLOORMODE);
    showInfo(sbOutMsgBuf);
}


/*
 * Function
 */
int
```

```
collGetSesFormatHandler(fd)
    int             fd;
{
    cmAckOk(fd);
    ShastraULongOut(fd, &pSesmFrontCD->lFormat);
    cmFlush(fd);

    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SESFORMAT);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
collSetSesFormatHandler(fd)
    int             fd;
{
    ShastraULongIn(fd, &pSesmFrontCD->lFormat);
    cmAckOk(fd);
    ShastraULongOut(fd, &pSesmFrontCD->lFormat);
    cmFlush(fd);

    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollSetSesFormatHandler,
                (char *) &pSesmFrontCD->lFormat);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_SESFORMAT);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
collGrabTokenHandler(fd)
    int             fd;
{
/*
    actual floor control processing, bcast if something changes
*/
    pSesmFrontCD->sIdTagToken = localShaIdIn[fd].lSIDTag;

    cmAckOk(fd);
    ShastraIdTagOut(fd, &pSesmFrontCD->sIdTagToken);
    cmFlush(fd);

    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollAskTokenHandler,
                (char *) &pSesmFrontCD->sIdTagToken);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GRAB_TOKEN);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
collFreeTokenHandler(fd)
    int             fd;
{
    pSesmFrontCD->sIdTagToken = pShastraFrontIdTags->shastraIdTags_val[0];
    cmAckOk(fd);
    cmFlush(fd);

    {
        int            *pfd;
        int             nfd;

        getKrFDsBCast(&pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollAskTokenHandler,
                (char *) &pSesmFrontCD->sIdTagToken);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_FREE_TOKEN);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
collTellTokenHandler(fd)
    int             fd;
{
    shastraIdTag    sIdTagToken;
    int outFd;

    ShastraIdTagIn(fd, &sIdTagToken);
    cmAckOk(fd);
    cmFlush(fd);
/*CHECK floor processing*/
    pSesmFrontCD->sIdTagToken = sIdTagToken;
    switch(routeFrontSIdTagToFd(&sIdTagToken, &outFd,
            "collTellTokenHandler()")){
        case route_FRONT:
            putCollGrabTokenHandler(outFd, &sIdTagToken);
        break;
        case route_ERROR:
        default:
        break;
    }

    {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(outFd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollAskTokenHandler,
                (char *) &pSesmFrontCD->sIdTagToken);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_TELL_TOKEN);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int
collAskTokenHandler(fd)
    int             fd;
{
    cmAckOk(fd);
    ShastraIdTagOut(fd, &pSesmFrontCD->sIdTagToken);
    cmFlush(fd);

    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_ASK_TOKEN);
    showInfo(sbOutMsgBuf);
}


/*
 * Function
 */

int
collStartPictHandler(fd)
    int             fd;
{
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    cmAckOk(fd);
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);

    pSesMgrCollData->fPictState = COMM_STARTED;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
              & kernelShastraId.lSIDTag, pSIdTag);
    if (pCollFrData != NULL) {
        pCollFrData->fPictState = COMM_STARTED;
    } else {
        fprintf(stderr, "collStartPictHandler()->no SmFrData!");
    }
    {
        int             *pfd;
        int              nfd;
        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollStartPictHandler,
                (char *) &localShaIdIn[fd].lSIDTag);
    }

    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_START_PICT);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collEndPictHandler(fd)
    int             fd;
{
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    cmAckOk(fd);
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);

    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if (pCollFrData != NULL) {
        pCollFrData->fPictState = COMM_ENDED;
    } else {
        fprintf(stderr, "collStartPictHandler()->no SmFrData!");
    }
    {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollEndPictHandler,
                (char *) &localShaIdIn[fd].lSIDTag);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_END_PICT);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendPictHandler(fd)
```

```
    int             fd;
{
    char           *bufNam;
    bunchOfThings   bunch;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    bunch.nThings = 2;
    bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
    bunch.things[1] = bufNam;
    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollSendPictHandler,
                (char *) &bunch);
    }
    free(bufNam);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_PICT);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendMsgPictHandler(fd)
    int             fd;
{
    bunchOfThings   bunch;
    char           *buf;
    static pictPieces pictBites;
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    PictDataBitesIn(fd, &pictBites);
    cmAckOk(fd);
    cmFlush(fd);

    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
            & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fPictState == COMM_ENDED)) {
    } else {
        bunch.nThings = 2;
        bunch.things[0] = (char *) pSIdTag;
        bunch.things[1] = (char *) &pictBites;
        {
            int            *pfd;
            int             nfd;
```

```
                getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
                pSesMgrCollData->pShmInfoOut->shmDirty = 0;
                cmMultiCast(pfd, nfd, putCollSendMsgPictHandler,
                        (char *) &bunch);
                pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            }
        }
        return;
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGPICT);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */
    int
    collRecvdMsgPictHandler(fd)
        int             fd;
    {
        char           *bufNam;

        bufNam = cmReceiveString(fd);
        cmAckOk(fd);
        cmFlush(fd);

        free(bufNam);
        return;
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGPICT);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */
    int
    collSendMsgShmPictHandler(fd)
        int             fd;
    {
        int             shmId;
        bunchOfThings   bunch;
        static pictPieces pictBites;
        shastraIdTag   *pSIdTag;
        collabFrontData *pCollFrData;
        shmInfo        *pShmInfo;
        int             n;

        ShastraIntIn(fd, &shmId);
        cmAckOk(fd);
        cmFlush(fd);

        if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
            fprintf(stderr, "collSendMsgShmPictHandler()->no non-local SHM\n");
            return;
        }
        pShmInfo = mplexInShmInfo(fd);
    if (!shMemReconnect(pShmInfo, shmId)) {
        fprintf(stderr, "collSendMsgShmPictHandler()->SHM recon problem\n")
            ;
        return;
    }
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
                & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fPictState == COMM_ENDED)) {
    } else {

        pictPiecesMemIn(pShmInfo->shmAddr, pShmInfo->shmSize,
                   &pictBites);
        bunch.nThings = 2;
        bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
        bunch.things[1] = (char *) &pictBites;
        {
            int             *pfd;
            int              nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgPictHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
    return;
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMPICT);
    showInfo(sbOutMsgBuf);

}
/*
 * Function
 */
int
collRecvdMsgShmPictHandler(fd)
    int             fd;
{
    int             shmId;

    ShastraIntIn(fd, &shmId);
    cmAckOk(fd);
    cmFlush(fd);

    if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
        fprintf(stderr, "collRecvdMsgShmPictHandler()->no non-local SHM\n")
            ;
        return;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMPICT);
    showInfo(sbOutMsgBuf);
}
```

```
/*
 * Function
 */

int
collStartXSCntlHandler(fd)
    int             fd;
{
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    cmAckOk(fd);
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);

    pSesMgrCollData->fXSCntlState = COMM_STARTED;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if (pCollFrData != NULL) {
        pCollFrData->fXSCntlState = COMM_STARTED;
    } else {
        fprintf(stderr, "collStartXSCntlHandler()->no SmFrData!");
    }
    {
        int            *pfd;
        int             nfd;
        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollStartXSCntlHandler,
                (char *) &localShaIdIn[fd].lSIDTag);
    }

    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_START_XSCNTL);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collEndXSCntlHandler(fd)
    int             fd;
{
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    cmAckOk(fd);
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);

    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if (pCollFrData != NULL) {
        pCollFrData->fXSCntlState = COMM_ENDED;
    } else {
        fprintf(stderr, "collStartXSCntlHandler()->no SmFrData!");
    }
    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollEndXSCntlHandler,
                (char *) &localShaIdIn[fd].lSIDTag);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_END_XSCNTL);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendXSCntlHandler(fd)
    int             fd;
{
    char           *bufNam;
    bunchOfThings   bunch;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    bunch.nThings = 2;
    bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
    bunch.things[1] = bufNam;
    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollSendXSCntlHandler,
                (char *) &bunch);
    }
    free(bufNam);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_XSCNTL);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendMsgXSCntlHandler(fd)
    int             fd;
{
    bunchOfThings   bunch;
    char           *buf;
    static xsCntlDatas xsCntlBites;
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    XSCntlBitesIn(fd, &xsCntlBites);
    cmAckOk(fd);
    cmFlush(fd);

    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fXSCntlState == COMM_ENDED))
        {
    } else {
        bunch.nThings = 2;
        bunch.things[0] = (char *) pSIdTag;
        bunch.things[1] = (char *) &xsCntlBites;
        {
            int            *pfd;
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgXSCntlHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
    return;
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGXSCNTL);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collRecvdMsgXSCntlHandler(fd)
    int             fd;
{
    char           *bufNam;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    free(bufNam);
    return;
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGXSCNTL);
    showInfo(sbOutMsgBuf);
}
/*
```

```
 * Function
 */
int
collSendMsgShmXSCntlHandler(fd)
    int             fd;
{
    int             shmId;
    bunchOfThings   bunch;
    static xsCntlDatas xsCntlBites;
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;
    shmInfo        *pShmInfo;
    int             n;

    ShastraIntIn(fd, &shmId);
    cmAckOk(fd);
    cmFlush(fd);

    if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
        fprintf(stderr, "collSendMsgShmXSCntlHandler()->no non-local SHM\n"
            );
        return;
    }
    pShmInfo = mplexInShmInfo(fd);
    if (!shMemReconnect(pShmInfo, shmId)) {
        fprintf(stderr, "collSendMsgShmXSCntlHandler()->SHM recon problem\
            n");
        return;
    }
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
              & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fXSCntlState == COMM_ENDED))
        {
    } else {

        xsCntlDatasMemIn(pShmInfo->shmAddr, pShmInfo->shmSize,
                 &xsCntlBites);
        bunch.nThings = 2;
        bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
        bunch.things[1] = (char *) &xsCntlBites;
        {
            int             *pfd;
            int              nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgXSCntlHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
    return;
```

```
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMXSCNTL);
        showInfo(sbOutMsgBuf);

    }
    /*
     * Function
     */
    int
    collRecvdMsgShmXSCntlHandler(fd)
        int             fd;
    {
        int             shmId;

        ShastraIntIn(fd, &shmId);
        cmAckOk(fd);
        cmFlush(fd);

        if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
            fprintf(stderr, "collRecvdMsgShmXSCntlHandler()->no non-local SHM\
                n");
            return;
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMXSCNTL);
        showInfo(sbOutMsgBuf);
    }


    /*
     * Function
     */
    int
    collStartPolyHandler(fd)
        int             fd;
    {
        shastraIdTag   *pSIdTag;
        collabFrontData *pCollFrData;

        cmAckOk(fd);
        pSIdTag = & localShaIdIn[fd].lSIDTag;
        ShastraIdTagOut(fd, pSIdTag);
        cmFlush(fd);

        pSesMgrCollData->fPolyState = COMM_STARTED;
        pCollFrData = (collabFrontData *) getSesMgrFrontData(
                    & kernelShastraId.lSIDTag, pSIdTag);
        if (pCollFrData != NULL) {
            pCollFrData->fPolyState = COMM_STARTED;
        } else {
            fprintf(stderr, "collStartPolyHandler()->no SmFrData!");
        }
        {
            int             *pfd;
            int              nfd;
            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putCollStartPolyHandler,
                    (char *) &localShaIdIn[fd].lSIDTag);
        }

        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_START_POLY);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */
    int
    collEndPolyHandler(fd)
        int             fd;
    {
        shastraIdTag   *pSIdTag;
        collabFrontData *pCollFrData;

        cmAckOk(fd);
        pSIdTag = & localShaIdIn[fd].lSIDTag;
        ShastraIdTagOut(fd, pSIdTag);
        cmFlush(fd);

        pCollFrData = (collabFrontData *) getSesMgrFrontData(
                   & kernelShastraId.lSIDTag, pSIdTag);
        if (pCollFrData != NULL) {
            pCollFrData->fPolyState = COMM_ENDED;
        } else {
            fprintf(stderr, "collStartPolyHandler()->no SmFrData!");
        }
        {
            int            *pfd;
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putCollEndPolyHandler,
                    (char *) &localShaIdIn[fd].lSIDTag);
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_END_POLY);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */
    int
    collSendPolyHandler(fd)
        int             fd;
    {
        char           *bufNam;
        bunchOfThings   bunch;

        bufNam = cmReceiveString(fd);
        cmAckOk(fd);
    cmFlush(fd);

    bunch.nThings = 2;
    bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
    bunch.things[1] = bufNam;
    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollSendPolyHandler,
                (char *) &bunch);
    }
    free(bufNam);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_POLY);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendMsgPolyHandler(fd)
    int             fd;
{
    bunchOfThings   bunch;
    char           *buf;
    static ipimageData image;
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    ImageDataIn(fd, &image);
    cmAckOk(fd);
    cmFlush(fd);

    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fPolyState == COMM_ENDED)) {
    } else {
        bunch.nThings = 2;
        bunch.things[0] = (char *) pSIdTag;
        bunch.things[1] = (char *) &image;
        {
            int            *pfd;
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgPolyHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
    return;
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGPOLY);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collRecvdMsgPolyHandler(fd)
    int             fd;
{
    char           *bufNam;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    free(bufNam);
    return;
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGPOLY);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendMsgShmPolyHandler(fd)
    int             fd;
{
    int             shmId;
    bunchOfThings   bunch;
    static ipimageData image;
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;
    shmInfo        *pShmInfo;
    int             n;

    ShastraIntIn(fd, &shmId);
    cmAckOk(fd);
    cmFlush(fd);

    if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
        fprintf(stderr, "collSendMsgShmPolyHandler()->no non-local SHM\n");
        return;
    }
    pShmInfo = mplexInShmInfo(fd);
    if (!shMemReconnect(pShmInfo, shmId)) {
        fprintf(stderr, "collSendMsgShmPolyHandler()->SHM recon problem\n")
            ;
        return;
    }
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
                & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fPolyState == COMM_ENDED)) {
    } else {

        ipimageDataMemIn(pShmInfo->shmAddr, pShmInfo->shmSize,
                    &image);
        bunch.nThings = 2;
        bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
        bunch.things[1] = (char *) &image;
        {
            int             *pfd;
            int              nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgPolyHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
    return;
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMPOLY);
    showInfo(sbOutMsgBuf);

}
/*
 * Function
 */
int
collRecvdMsgShmPolyHandler(fd)
    int             fd;
{
    int             shmId;

    ShastraIntIn(fd, &shmId);
    cmAckOk(fd);
    cmFlush(fd);

    if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
        fprintf(stderr, "collRecvdMsgShmPolyHandler()->no non-local SHM\n")
            ;
        return;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMPOLY);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */

int
collStartPntrHandler(fd)
```

```
    int             fd;
{
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    cmAckOk(fd);
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);

    pSesMgrCollData->fPntrState = COMM_STARTED;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
                & kernelShastraId.lSIDTag, pSIdTag);
    if (pCollFrData != NULL) {
        pCollFrData->fPntrState = COMM_STARTED;
    } else {
        fprintf(stderr, "collStartPntrHandler()->no SmFrData!");
    }
    {
        int             *pfd;
        int              nfd;
        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollStartPntrHandler,
                (char *) &localShaIdIn[fd].lSIDTag);
    }

    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_START_PNTR);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collEndPntrHandler(fd)
    int             fd;
{
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    cmAckOk(fd);
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);

    pCollFrData = (collabFrontData *) getSesMgrFrontData(
                & kernelShastraId.lSIDTag, pSIdTag);
    if (pCollFrData != NULL) {
        pCollFrData->fPntrState = COMM_ENDED;
    } else {
        fprintf(stderr, "collStartPntrHandler()->no SmFrData!");
    }
    {
        int             *pfd;
```

```
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putCollEndPntrHandler,
                    (char *) &localShaIdIn[fd].lSIDTag);
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_END_PNTR);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */
    int
    collSendPntrHandler(fd)
        int             fd;
    {
        char           *bufNam;
        bunchOfThings   bunch;

        bufNam = cmReceiveString(fd);
        cmAckOk(fd);
        cmFlush(fd);

        bunch.nThings = 2;
        bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
        bunch.things[1] = bufNam;
        {
            int            *pfd;
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putCollSendPntrHandler,
                    (char *) &bunch);
        }
        free(bufNam);
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_PNTR);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */
    int
    collSendMsgPntrHandler(fd)
        int             fd;
    {
        bunchOfThings   bunch;
        static shaDoubles pntrData;
        shastraIdTag   *pSIdTag;
        collabFrontData *pCollFrData;

        PntrBiteIn(fd, &pntrData);
        cmAckOk(fd);
        cmFlush(fd);
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
                & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fPntrState == COMM_ENDED)) {
    } else {
        bunch.nThings = 2;
        bunch.things[0] = (char *) pSIdTag;
        bunch.things[1] = (char *) &pntrData;
        {
            int             *pfd;
            int              nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgPntrHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGPNTR);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collRecvdMsgPntrHandler(fd)
    int             fd;
{
    char            *bufNam;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    free(bufNam);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGPNTR);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendMsgShmPntrHandler(fd)
    int             fd;
{
    int             shmId;
    bunchOfThings   bunch;
    char            *buf;
    shastraIdTag    *pSIdTag;
    collabFrontData *pCollFrData;
    shmInfo         *pShmInfo;
    int                 n;

    ShastraIntIn(fd, &shmId);
    cmAckOk(fd);
    cmFlush(fd);

    if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
        fprintf(stderr, "collSendMsgShmPntrHandler()->no non-local SHM\n");
        return;
    }
    pShmInfo = mplexInShmInfo(fd);
    if (!shMemReconnect(pShmInfo, shmId)) {
        fprintf(stderr, "collSendMsgShmPntrHandler()->SHM recon problem\n")
            ;
        return;
    }
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fPntrState == COMM_ENDED)) {
    } else {
        buf = pShmInfo->shmAddr;
        bunch.nThings = 2;
        bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
        bunch.things[1] = buf;
        {
            int            *pfd;
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgPntrHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMPNTR);
    showInfo(sbOutMsgBuf);

}
/*
 * Function
 */
int
collRecvdMsgShmPntrHandler(fd)
    int                 fd;
{
    int                 shmId;

    ShastraIntIn(fd, &shmId);
    cmAckOk(fd);
    cmFlush(fd);
```

```
        if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
            fprintf(stderr, "collRecvdMsgShmPntrHandler()->no non-local SHM\n")
                ;
            return;
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMPNTR);
        showInfo(sbOutMsgBuf);
    }



    /*
     * Function
     */

    int
    collStartCursorHandler(fd)
        int             fd;
    {
        shastraIdTag   *pSIdTag;
        collabFrontData *pCollFrData;

        cmAckOk(fd);
        pSIdTag = & localShaIdIn[fd].lSIDTag;
        ShastraIdTagOut(fd, pSIdTag);
        cmFlush(fd);

        pSesMgrCollData->fCursorState = COMM_STARTED;
        pCollFrData = (collabFrontData *) getSesMgrFrontData(
                    & kernelShastraId.lSIDTag, pSIdTag);
        if (pCollFrData != NULL) {
            pCollFrData->fCursorState = COMM_STARTED;
        } else {
            fprintf(stderr, "collStartCursorHandler()->no SmFrData!");
        }
        {
            int             *pfd;
            int              nfd;
            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putCollStartCursorHandler,
                    (char *) &localShaIdIn[fd].lSIDTag);
        }

        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_START_CURSOR);
        showInfo(sbOutMsgBuf);
    }
    /*
     * Function
     */
    int
    collEndCursorHandler(fd)
        int             fd;
    {
```

```
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    cmAckOk(fd);
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);

    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if (pCollFrData != NULL) {
        pCollFrData->fCursorState = COMM_ENDED;
    } else {
        fprintf(stderr, "collStartCursorHandler()->no SmFrData!");
    }
    {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollEndCursorHandler,
                (char *) &localShaIdIn[fd].lSIDTag);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_END_CURSOR);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendCursorHandler(fd)
    int             fd;
{
    char           *bufNam;
    bunchOfThings   bunch;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    bunch.nThings = 2;
    bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
    bunch.things[1] = bufNam;
    {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putCollSendCursorHandler,
                (char *) &bunch);
    }
    free(bufNam);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_CURSOR);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendMsgCursorHandler(fd)
    int             fd;
{
    bunchOfThings   bunch;
    static shaDoubles pntrData;
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;

    CursorBiteIn(fd, &pntrData);
    cmAckOk(fd);
    cmFlush(fd);

    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fCursorState == COMM_ENDED))
        {
    } else {
        bunch.nThings = 2;
        bunch.things[0] = (char *) pSIdTag;
        bunch.things[1] = (char *) &pntrData;
        {
            int            *pfd;
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgCursorHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGCURSOR);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collRecvdMsgCursorHandler(fd)
    int             fd;
{
    char           *bufNam;

    bufNam = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);
    free(bufNam);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGCURSOR);
    showInfo(sbOutMsgBuf);
}
/*
 * Function
 */
int
collSendMsgShmCursorHandler(fd)
    int             fd;
{
    int             shmId;
    bunchOfThings   bunch;
    char           *buf;
    shastraIdTag   *pSIdTag;
    collabFrontData *pCollFrData;
    shmInfo        *pShmInfo;
    int             n;

    ShastraIntIn(fd, &shmId);
    cmAckOk(fd);
    cmFlush(fd);

    if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
        fprintf(stderr, "collSendMsgShmCursorHandler()->no non-local SHM\n"
            );
        return;
    }
    pShmInfo = mplexInShmInfo(fd);
    if (!shMemReconnect(pShmInfo, shmId)) {
        fprintf(stderr, "collSendMsgShmCursorHandler()->SHM recon problem\
            n");
        return;
    }
    pSIdTag = & localShaIdIn[fd].lSIDTag;
    pCollFrData = (collabFrontData *) getSesMgrFrontData(
               & kernelShastraId.lSIDTag, pSIdTag);
    if ((pCollFrData == NULL) || (pCollFrData->fCursorState == COMM_ENDED))
        {
    } else {
        buf = pShmInfo->shmAddr;
        bunch.nThings = 2;
        bunch.things[0] = (char *) &localShaIdIn[fd].lSIDTag;
        bunch.things[1] = buf;
        {
            int            *pfd;
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
            cmMultiCast(pfd, nfd, putCollSendMsgCursorHandler,
                    (char *) &bunch);
            pSesMgrCollData->pShmInfoOut->shmDirty = 0;
        }
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMCURSOR);
    showInfo(sbOutMsgBuf);

}
/*
 * Function
 */
int
collRecvdMsgShmCursorHandler(fd)
    int             fd;
{
    int             shmId;

    ShastraIntIn(fd, &shmId);
    cmAckOk(fd);
    cmFlush(fd);

    if (kernelShastraId.lIPAddr != localShaIdIn[fd].lIPAddr) {
        fprintf(stderr, "collRecvdMsgShmCursorHandler()->no non-local SHM\
            n");
        return;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMCURSOR);
    showInfo(sbOutMsgBuf);
}




/*
 * Function
 */
int
putCollTellLeaderHandler(fd, pSIdTagSesm, pSIdTagLdr, pIdTag)
    int             fd;
    shastraIdTag   *pSIdTagSesm;
    shastraIdTag   *pSIdTagLdr;
    unsigned long  *pIdTag;
{
    putStringOnChannel(fd, REQ_COLL_TELLLEADER, "putCollTellLeaderHandler(
        )");
    ShastraIdTagOut(fd, pSIdTagSesm);
    ShastraIdTagOut(fd, pSIdTagLdr);
    ShastraULongOut(fd, pIdTag);
    cmFlush(fd);

    if (debug) {
        outputIdTag(stderr, pSIdTagSesm);
        outputIdTag(stderr, pSIdTagLdr);
    }
    }


    /*
     * Function
     */
    int
    putShaSesmFrIdHandler(fd, pSIdTagSesm)
        int             fd;
        shastraIdTag   *pSIdTagSesm;
    {
        shastraIdTags  *pSIdTags;

        putStringOnChannel(fd, REQ_SET_SHASESMFRID, "putShaSesmFrIdHandler()");
        pSIdTags = getSesmFrontSIdTags(pSIdTagSesm);
        ShastraIdTagOut(fd, pSIdTagSesm);
        ShastraIdTagsOut(fd, pSIdTags);
        cmFlush(fd);

        if (debug) {
            outputIdTag(stderr, pSIdTagSesm);
            outputIdTags(stderr, pSIdTags);
        }
    }

    /*
     * Function
     */
    int
    putCollLeaveHandler(fd)
        int             fd;
    {
        putStringOnChannel(fd, REQ_COLL_LEAVE, "putCollLeaveHandler()");
        cmFlush(fd);
    }

    /*
     * Function
     */
    int
    putCollAskJoinHandler(fd, pSmSIdTag, pSIdTag)
        int             fd;
        shastraIdTag   *pSIdTag;
        shastraIdTag   *pSmSIdTag;
    {
        putStringOnChannel(fd, REQ_COLL_ASKJOIN, "putCollAskJoinHandler()");
        ShastraIdTagOut(fd, pSmSIdTag);
        ShastraIdTagOut(fd, pSIdTag);
        cmFlush(fd);
    }

    /*
     * Function
```

```
 */
int putCollAskJoinMsgHandler(fd, pSmSIdTag, pSIdTag, sbMsg)
    int fd;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COLL_ASKJOINMSG, "putCollAskJoinMsgHandler(
        )");
    ShastraIdTagOut(fd, pSmSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
}

/*
 * Function
 */
int putCollAskJnRespMsgHandler(fd, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
    int fd;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COLL_ASKJNRESPMSG,
        "putCollAskJnRespMsgHandler()");
    ShastraIdTagOut(fd, pSmSIdTag);
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
}

/*
 * Function
 */
int putCollAskJnStatusHandler(fd, pSmSIdTag, pToSIdTag, pSIdTag, lStatus)
    int fd;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    shaULong lStatus;
{
    putStringOnChannel(fd, REQ_COLL_ASKJNSTATUS, "putCollAskJnStatusHandler
        ()");
    ShastraIdTagOut(fd, pSmSIdTag);
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    ShastraULongOut(fd, &lStatus);
    cmFlush(fd);
}
```

```
/*
 * Function
 */
int
putCollTellJoinHandler(fd, pSmSIdTag, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
    shastraIdTag   *pSmSIdTag;
{
    putStringOnChannel(fd, REQ_COLL_TELLJOIN, "putCollTellJoinHandler()");
    ShastraIdTagOut(fd, pSmSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}

/*
 * Function
 */

int
putCollStartTextHandler(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    putStringOnChannel(fd, REQ_START_TEXT, "putCollStartTextHandler()");
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollEndTextHandler(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    putStringOnChannel(fd, REQ_END_TEXT, "putCollEndTextHandler()");
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollSendTextHandler(fd, buf)
    int             fd;
    char           *buf;
{
    bunchOfThings  *bunch;
    bunch = (bunchOfThings *) buf;
    putStringOnChannel(fd, REQ_SEND_TEXT, "putCollSendTextHandler()");
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    putStringOnChannel(fd, bunch->things[1], "putCollSendTextHandler()");
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollSendMsgTextHandler(fd, buf)
    int                 fd;
    char               *buf;
{
    bunchOfThings      *bunch;
    char               *msg;
    int                 n;
    shmInfo            *pShmInfo;

    bunch = (bunchOfThings *) buf;
    msg = bunch->things[1];
#ifdef USESHAREDMEMFORTEXT
    if (kernelShastraId.lIPAddr == localShaIdIn[fd].lIPAddr) {
        pShmInfo = pSesMgrCollData->pShmInfoOut;
        if (!pShmInfo->shmDirty) {
            pShmInfo->shmDirty = 1;
            n = strlen(msg) + 1;
            if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0
                ) {
                fprintf(stderr, "putCollSendMsgTextHandler()->couldn't
                    shMemReuseSegment!\n");
            }
            memcpy(pShmInfo->shmAddr, msg, n);
        }
        putStringOnChannel(fd, REQ_SEND_MSGSHMTEXT,
            "putCollSendMsgTextHandler()");
        ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
        ShastraIntOut(fd, &pShmInfo->shmId);
        cmFlush(fd);
        return;
    }
#endif                  /* USESHAREDMEMFORTEXT */

    putStringOnChannel(fd, REQ_SEND_MSGTEXT, "putCollSendMsgTextHandler()")
        ;
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    putStringOnChannel(fd, bunch->things[1], "putCollSendMsgTextHandler()")
        ;
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollRecvdMsgTextHandler(fd, buf)
    int                 fd;
    char               *buf;
```

```
{
    putStringOnChannel(fd, REQ_RECVD_MSGTEXT, "putCollRecvdMsgTextHandler(
        )");
    putStringOnChannel(fd, buf, "putCollRecvdMsgTextHandler()");
    cmFlush(fd);
}

/*
 * Function
 */

int
putCollStartAudioHandler(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    putStringOnChannel(fd, REQ_START_AUDIO, "putCollStartAudioHandler()");
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollEndAudioHandler(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    putStringOnChannel(fd, REQ_END_AUDIO, "putCollEndAudioHandler()");
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollSendAudioHandler(fd, buf)
    int             fd;
    char           *buf;
{
    bunchOfThings  *bunch;
    bunch = (bunchOfThings *) buf;
    putStringOnChannel(fd, REQ_SEND_AUDIO, "putCollSendAudioHandler()");
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    putStringOnChannel(fd, bunch->things[1], "putCollSendAudioHandler()");
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollSendMsgAudioHandler(fd, buf)
    int             fd;
    char            *buf;
{
    bunchOfThings  *bunch;
    audioBite      *pABite;
    int             n;
    shmInfo        *pShmInfo;

    bunch = (bunchOfThings *) buf;
    pABite = (audioBite *) bunch->things[1];
#ifdef USESHAREDMEMFORAUDIO
    if (kernelShastraId.lIPAddr == localShaIdIn[fd].lIPAddr) {
        pShmInfo = pSesMgrCollData->pShmInfoOut;
        if (!pShmInfo->shmDirty) {
            pShmInfo->shmDirty = 1;
            n = pABite->data.data_len + sizeof(audioBite);
            if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0
                ) {
                fprintf(stderr, "putCollSendMsgAudioHandler()->couldn't
                    shMemReuseSegment!\n");
            }
            audioBiteMemOut(pShmInfo->shmAddr, pShmInfo->shmSize, pABite);
        }
        putStringOnChannel(fd, REQ_SEND_MSGSHMAUDIO,
                           "putCollSendMsgAudioHandler()");
        ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
        ShastraIntOut(fd, &pShmInfo->shmId);
        cmFlush(fd);
        return;
    }
#endif              /* USESHAREDMEMFORAUDIO */
    putStringOnChannel(fd, REQ_SEND_MSGAUDIO, "putCollSendMsgAudioHandler(
        )");
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    AudioBiteOut(fd, pABite);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollRecvdMsgAudioHandler(fd, buf)
    int             fd;
    char           *buf;
{
    putStringOnChannel(fd, REQ_RECVD_MSGAUDIO, "putCollRecvdMsgAudioHandler
        ()");
    putStringOnChannel(fd, buf, "putCollRecvdMsgAudioHandler()");
    cmFlush(fd);
}

/*
 * Function
 */
```

```
int
putCollStartVideoHandler(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    putStringOnChannel(fd, REQ_START_VIDEO, "putCollStartVideoHandler()");
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollEndVideoHandler(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    putStringOnChannel(fd, REQ_END_VIDEO, "putCollEndVideoHandler()");
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollSendVideoHandler(fd, buf)
    int             fd;
    char           *buf;
{
    bunchOfThings  *bunch;
    bunch = (bunchOfThings *) buf;
    putStringOnChannel(fd, REQ_SEND_VIDEO, "putCollSendVideoHandler()");
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    putStringOnChannel(fd, bunch->things[1], "putCollSendVideoHandler()");
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollSendMsgVideoHandler(fd, buf)
    int             fd;
    char           *buf;
{
    bunchOfThings  *bunch;
    videoImg       *pVImg;
    int             n;
    shmInfo        *pShmInfo;

    bunch = (bunchOfThings *) buf;
    pVImg = (videoImg *) bunch->things[1];
#ifdef USESHAREDMEM
```

```
    if (kernelShastraId.lIPAddr == localShaIdIn[fd].lIPAddr) {
        pShmInfo = pSesMgrCollData->pShmInfoOut;
        if (!pShmInfo->shmDirty) {
            pShmInfo->shmDirty = 1;
            n = pVImg->data.data_len + sizeof(videoImg);
            if (shMemReuseSegment(pShmInfo, ((n > 102400) ? n : 102400)) ==
                0) {
                fprintf(stderr, "putCollSendMsgVideoHandler()->couldn't
                    shMemReuseSegment!\n");
            }
            videoImgMemOut(pShmInfo->shmAddr, pShmInfo->shmSize, pVImg);
        }
        putStringOnChannel(fd, REQ_SEND_MSGSHMVIDEO,
                    "putCollSendMsgVideoHandler()");
        ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
        ShastraIntOut(fd, &pShmInfo->shmId);
        cmFlush(fd);
        return;
    }
#endif              /* USESHAREDMEM */

    bunch = (bunchOfThings *) buf;
    putStringOnChannel(fd, REQ_SEND_MSGVIDEO, "putCollSendMsgVideoHandler(
        )");
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    pVImg = (videoImg *) bunch->things[1];
    VideoImgOut(fd, pVImg);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollRecvdMsgVideoHandler(fd, buf)
    int             fd;
    char           *buf;
{
    putStringOnChannel(fd, REQ_RECVD_MSGVIDEO, "putCollRecvdMsgVideoHandler
        ()");
    putStringOnChannel(fd, buf, "putCollRecvdMsgVideoHandler()");
    cmFlush(fd);
}

/*
 * Function
 */

int
putCollStartPolyHandler(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    putStringOnChannel(fd, REQ_START_POLY, "putCollStartPolyHandler()");
```

```
        ShastraIdTagOut(fd, pSIdTag);
        cmFlush(fd);
    }
    /*
     * Function
     */
    int
    putCollEndPolyHandler(fd, pSIdTag)
        int             fd;
        shastraIdTag   *pSIdTag;
    {
        putStringOnChannel(fd, REQ_END_POLY, "putCollEndPolyHandler()");
        ShastraIdTagOut(fd, pSIdTag);
        cmFlush(fd);
    }
    /*
     * Function
     */
    int
    putCollSendPolyHandler(fd, buf)
        int             fd;
        char           *buf;
    {
        bunchOfThings  *bunch;
        bunch = (bunchOfThings *) buf;
        putStringOnChannel(fd, REQ_SEND_POLY, "putCollSendPolyHandler()");
        ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
        putStringOnChannel(fd, bunch->things[1], "putCollSendPolyHandler()");
        cmFlush(fd);
    }
    /*
     * Function
     */
    int
    putCollSendMsgPolyHandler(fd, buf)
        int             fd;
        char           *buf;
    {
        bunchOfThings  *bunch;
        ipimageData    *pImage;
        int             n;
        shmInfo        *pShmInfo;

        bunch = (bunchOfThings *) buf;
        pImage = (ipimageData *) bunch->things[1];
    #ifdef USESHAREDMEMFORMPOLY
        if (kernelShastraId.lIPAddr == localShaIdIn[fd].lIPAddr) {
            pShmInfo = pSesMgrCollData->pShmInfoOut;
            if (!pShmInfo->shmDirty) {
                pShmInfo->shmDirty = 1;
                n = pImage->mPoly->nPolygons * 100 * sizeof(double);
                if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0
                    ) {
```

```
                    fprintf(stderr, "putCollSendMsgPolyHandler()->couldn't
                        shMemReuseSegment!\n");
                }
                ipimageDataMemOut(pShmInfo->shmAddr, pShmInfo->shmSize, pImage)
                    ;
            }
            putStringOnChannel(fd, REQ_SEND_MSGSHMPOLY,
                        "putCollSendMsgPolyHandler()");
            ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
            ShastraIntOut(fd, &pShmInfo->shmId);
            cmFlush(fd);
            return;
        }
#endif              /* USESHAREDMEMFORMPOLY */
    putStringOnChannel(fd, REQ_SEND_MSGPOLY, "putCollSendMsgPolyHandler()")
        ;
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    ImageDataOut(fd, pImage);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollRecvdMsgPolyHandler(fd, buf)
    int             fd;
    char           *buf;
{
    putStringOnChannel(fd, REQ_RECVD_MSGPOLY, "putCollRecvdMsgPolyHandler(
        )");
    putStringOnChannel(fd, buf, "putCollRecvdMsgPolyHandler()");
    cmFlush(fd);
}

/*
 * Function
 */

int
putCollStartPictHandler(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    putStringOnChannel(fd, REQ_START_PICT, "putCollStartPictHandler()");
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollEndPictHandler(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    putStringOnChannel(fd, REQ_END_PICT, "putCollEndPictHandler()");
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollSendPictHandler(fd, buf)
    int             fd;
    char           *buf;
{
    bunchOfThings  *bunch;
    bunch = (bunchOfThings *) buf;
    putStringOnChannel(fd, REQ_SEND_PICT, "putCollSendPictHandler()");
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    putStringOnChannel(fd, bunch->things[1], "putCollSendPictHandler()");
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollSendMsgPictHandler(fd, buf)
    int             fd;
    char           *buf;
{
    bunchOfThings  *bunch;
    pictPieces  *pPCBites;
    int             n;
    shmInfo        *pShmInfo;

    bunch = (bunchOfThings *) buf;
    pPCBites = (pictPieces *) bunch->things[1];
#ifdef USESHAREDMEMFORPICT
    if (kernelShastraId.lIPAddr == localShaIdIn[fd].lIPAddr) {
        pShmInfo = pSesMgrCollData->pShmInfoOut;
        if (!pShmInfo->shmDirty) {
            pShmInfo->shmDirty = 1;
/*CHECK*/
            n = 0;
            if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0
                ) {
                fprintf(stderr, "putCollSendMsgPictHandler()->couldn't
                    shMemReuseSegment!\n");
            }
            pictPiecesMemOut(pShmInfo->shmAddr, pShmInfo->shmSize, pPCBites
                );
        }
        putStringOnChannel(fd, REQ_SEND_MSGSHMPICT,
                   "putCollSendMsgPictHandler()");
        ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
        ShastraIntOut(fd, &pShmInfo->shmId);
        cmFlush(fd);
        return;
    }
#endif              /* USESHAREDMEMFORPICT */
    putStringOnChannel(fd, REQ_SEND_MSGPICT, "putCollSendMsgPictHandler()")
        ;
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    PictDataBitesOut(fd, pPCBites);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollRecvdMsgPictHandler(fd, buf)
    int                fd;
    char              *buf;
{
    putStringOnChannel(fd, REQ_RECVD_MSGPICT, "putCollRecvdMsgPictHandler(
        )");
    putStringOnChannel(fd, buf, "putCollRecvdMsgPictHandler()");
    cmFlush(fd);
}

/*
 * Function
 */

int
putCollStartXSCntlHandler(fd, pSIdTag)
    int                fd;
    shastraIdTag      *pSIdTag;
{
    putStringOnChannel(fd, REQ_START_XSCNTL, "putCollStartXSCntlHandler()")
        ;
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollEndXSCntlHandler(fd, pSIdTag)
    int                fd;
    shastraIdTag      *pSIdTag;
{
    putStringOnChannel(fd, REQ_END_XSCNTL, "putCollEndXSCntlHandler()");
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
```

```
 * Function
 */
int
putCollSendXSCntlHandler(fd, buf)
    int                 fd;
    char               *buf;
{
    bunchOfThings  *bunch;
    bunch = (bunchOfThings *) buf;
    putStringOnChannel(fd, REQ_SEND_XSCNTL, "putCollSendXSCntlHandler()");
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    putStringOnChannel(fd, bunch->things[1], "putCollSendXSCntlHandler()");
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollSendMsgXSCntlHandler(fd, buf)
    int                 fd;
    char               *buf;
{
    bunchOfThings  *bunch;
    xsCntlDatas    *pXSCBites;
    int                 n;
    shmInfo        *pShmInfo;

    bunch = (bunchOfThings *) buf;
    pXSCBites = (xsCntlDatas *) bunch->things[1];
#ifdef USESHAREDMEMFORXSCD
    if (kernelShastraId.lIPAddr == localShaIdIn[fd].lIPAddr) {
        pShmInfo = pSesMgrCollData->pShmInfoOut;
        if (!pShmInfo->shmDirty) {
            pShmInfo->shmDirty = 1;
/*CHECK*/
            n = 0;
            if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0
                ) {
                fprintf(stderr, "putCollSendMsgXSCntlHandler()->couldn't
                    shMemReuseSegment!\n");
            }
            xsCntlDatasMemOut(pShmInfo->shmAddr, pShmInfo->shmSize,
                pXSCBites);
        }
        putStringOnChannel(fd, REQ_SEND_MSGSHMXSCNTL,
                   "putCollSendMsgXSCntlHandler()");
        ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
        ShastraIntOut(fd, &pShmInfo->shmId);
        cmFlush(fd);
        return;
    }
#endif              /* USESHAREDMEMFORXSCD */
    putStringOnChannel(fd, REQ_SEND_MSGXSCNTL, "putCollSendMsgXSCntlHandler
        ()");
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    XSCntlBitesOut(fd, pXSCBites);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollRecvdMsgXSCntlHandler(fd, buf)
    int                fd;
    char              *buf;
{
    putStringOnChannel(fd, REQ_RECVD_MSGXSCNTL,
        "putCollRecvdMsgXSCntlHandler()");
    putStringOnChannel(fd, buf, "putCollRecvdMsgXSCntlHandler()");
    cmFlush(fd);
}

/*
 * Function
 */

int
putCollStartPntrHandler(fd, pSIdTag)
    int                fd;
    shastraIdTag      *pSIdTag;
{
    putStringOnChannel(fd, REQ_START_PNTR, "putCollStartPntrHandler()");
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollEndPntrHandler(fd, pSIdTag)
    int                fd;
    shastraIdTag      *pSIdTag;
{
    putStringOnChannel(fd, REQ_END_PNTR, "putCollEndPntrHandler()");
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollSendPntrHandler(fd, buf)
    int                fd;
    char              *buf;
{
    bunchOfThings     *bunch;
    bunch = (bunchOfThings *) buf;
    putStringOnChannel(fd, REQ_SEND_PNTR, "putCollSendPntrHandler()");
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    putStringOnChannel(fd, bunch->things[1], "putCollSendPntrHandler()");
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollSendMsgPntrHandler(fd, buf)
    int                 fd;
    char               *buf;
{
    bunchOfThings      *bunch;
    shaDoubles         *pPntrD;
    int                 n;
    shmInfo            *pShmInfo;

    bunch = (bunchOfThings *) buf;
    pPntrD = (shaDoubles *) bunch->things[1];
#ifdef USESHAREDMEMFORPNTR
    if (kernelShastraId.lIPAddr == localShaIdIn[fd].lIPAddr) {
        pShmInfo = pSesMgrCollData->pShmInfoOut;
        if (!pShmInfo->shmDirty) {
            pShmInfo->shmDirty = 1;
            n = strlen(msg) + 1;
            if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0
                ) {
                fprintf(stderr, "putCollSendMsgPntrHandler()->couldn't
                    shMemReuseSegment!\n");
            }
            memcpy(pShmInfo->shmAddr, msg, n);
        }
        putStringOnChannel(fd, REQ_SEND_MSGSHMPNTR,
            "putCollSendMsgPntrHandler()");
        ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
        ShastraIntOut(fd, &pShmInfo->shmId);
        cmFlush(fd);
        return;
    }
#endif                  /* USESHAREDMEMFORPNTR */

    putStringOnChannel(fd, REQ_SEND_MSGPNTR, "putCollSendMsgPntrHandler()")
        ;
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    PntrBiteOut(fd, pPntrD);
    cmFlush(fd);
}
/*
 * Function
 */
int
```

```
putCollRecvdMsgPntrHandler(fd, buf)
    int             fd;
    char           *buf;
{
    putStringOnChannel(fd, REQ_RECVD_MSGPNTR, "putCollRecvdMsgPntrHandler(
        )");
    putStringOnChannel(fd, buf, "putCollRecvdMsgPntrHandler()");
    cmFlush(fd);
}


/*
 * Function
 */

int
putCollStartCursorHandler(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    putStringOnChannel(fd, REQ_START_CURSOR, "putCollStartCursorHandler()")
        ;
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollEndCursorHandler(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    putStringOnChannel(fd, REQ_END_CURSOR, "putCollEndCursorHandler()");
    ShastraIdTagOut(fd, pSIdTag);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollSendCursorHandler(fd, buf)
    int             fd;
    char           *buf;
{
    bunchOfThings  *bunch;
    bunch = (bunchOfThings *) buf;
    putStringOnChannel(fd, REQ_SEND_CURSOR, "putCollSendCursorHandler()");
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    putStringOnChannel(fd, bunch->things[1], "putCollSendCursorHandler()");
    cmFlush(fd);
}
/*
```

```
 * Function
 */
int
putCollSendMsgCursorHandler(fd, buf)
    int             fd;
    char           *buf;
{
    bunchOfThings  *bunch;
    shaDoubles     *pCursorD;
    int             n;
    shmInfo        *pShmInfo;

    bunch = (bunchOfThings *) buf;
    pCursorD = (shaDoubles *) bunch->things[1];
#ifdef USESHAREDMEMFORCURSOR
    if (kernelShastraId.lIPAddr == localShaIdIn[fd].lIPAddr) {
        pShmInfo = pSesMgrCollData->pShmInfoOut;
        if (!pShmInfo->shmDirty) {
            pShmInfo->shmDirty = 1;
            n = strlen(msg) + 1;
            if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0
                ) {
                fprintf(stderr, "putCollSendMsgCursorHandler()->couldn't
                    shMemReuseSegment!\n");
            }
            memcpy(pShmInfo->shmAddr, msg, n);
        }
        putStringOnChannel(fd, REQ_SEND_MSGSHMCURSOR,
            "putCollSendMsgCursorHandler()");
        ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
        ShastraIntOut(fd, &pShmInfo->shmId);
        cmFlush(fd);
        return;
    }
#endif              /* USESHAREDMEMFORCURSOR */

    putStringOnChannel(fd, REQ_SEND_MSGCURSOR, "putCollSendMsgCursorHandler
        ()");
    ShastraIdTagOut(fd, (shastraIdTag *) bunch->things[0]);
    CursorBiteOut(fd, pCursorD);
    cmFlush(fd);
}
/*
 * Function
 */
int
putCollRecvdMsgCursorHandler(fd, buf)
    int             fd;
    char           *buf;
{
    putStringOnChannel(fd, REQ_RECVD_MSGCURSOR,
        "putCollRecvdMsgCursorHandler()");
    putStringOnChannel(fd, buf, "putCollRecvdMsgCursorHandler()");
```

```
        cmFlush(fd);
    }



    /*
     * Function
     */
    int
    putSetCollPermsHandler(fd, arg)
        int              fd;
        char            *arg;
    {
        shastraIdTag   *pSIdTag;
        shastraIdTag   *pPermTag;
        bunchOfThings  *bunch = (bunchOfThings *) arg;

        pSIdTag = (shastraIdTag *) bunch->things[0];
        pPermTag = (shastraIdTag *) bunch->things[1];
        putStringOnChannel(fd, REQ_SET_COLLPERMS, "putSetCollPermsHandler()");
        ShastraIdTagOut(fd, pSIdTag);
        ShastraIdTagOut(fd, pPermTag);
        cmFlush(fd);
    }

    /*
     * Function
     */
    int
    putSetSesmCollPermsHandler(fd, arg)
        int              fd;
        char            *arg;
    {
        shastraIdTag   *pSIdTag;
        shastraIdTags  *pPermTags;
        bunchOfThings  *bunch = (bunchOfThings *) arg;

        pSIdTag = (shastraIdTag *) bunch->things[0];
        pPermTags = (shastraIdTags *) bunch->things[1];
        putStringOnChannel(fd, REQ_SET_SESMCOLLPERMS,
            "putSetSesmCollPermsHandler()");
        ShastraIdTagOut(fd, pSIdTag);
        ShastraIdTagsOut(fd, pPermTags);
        cmFlush(fd);
    }

    /*
     * Function
     */
    int
    putCollSetIxnModeHandler(fd, pIxnMode)
        int              fd;
        unsigned long   *pIxnMode;
```

```
{
    putStringOnChannel(fd, REQ_SET_IXNMODE, "putCollSetIxnModeHandler()");
    ShastraULongOut(fd, pIxnMode);
    cmFlush(fd);
}

/*
 * Function
 */
int
putCollSetFloorModeHandler(fd, pFloorMode)
    int             fd;
    unsigned long  *pFloorMode;
{
    putStringOnChannel(fd, REQ_SET_FLOORMODE, "putCollSetFloorModeHandler(
        )");
    ShastraULongOut(fd, pFloorMode);
    cmFlush(fd);
}

/*
 * Function
 */
int
putCollSetSesFormatHandler(fd, pSesFormat)
    int             fd;
    unsigned long  *pSesFormat;
{
    putStringOnChannel(fd, REQ_SET_SESFORMAT, "putCollSetSesFormatHandler(
        )");
    ShastraULongOut(fd, pSesFormat);
    cmFlush(fd);
}

/*
 * Function
 */
int
putCollGrabTokenHandler(fd, pSIdTagToken)
    int             fd;
    shastraIdTag   *pSIdTagToken;
{
    putStringOnChannel(fd, REQ_GRAB_TOKEN, "putCollGrabTokenHandler()");
    ShastraIdTagOut(fd, pSIdTagToken);
    cmFlush(fd);
}

/*
 * Function
 */
int
putCollFreeTokenHandler(fd, pSIdTagToken)
    int             fd;
    shastraIdTag   *pSIdTagToken;
{
    putStringOnChannel(fd, REQ_FREE_TOKEN, "putCollFreeTokenHandler()");
    ShastraIdTagOut(fd, pSIdTagToken);
    cmFlush(fd);
}

/*
 * Function
 */
int
putCollTellTokenHandler(fd, pSIdTagToken)
    int             fd;
    shastraIdTag   *pSIdTagToken;
{
    putStringOnChannel(fd, REQ_TELL_TOKEN, "putCollTellTokenHandler()");
    ShastraIdTagOut(fd, pSIdTagToken);
    cmFlush(fd);
}

/*
 * Function
 */
int
putCollAskTokenHandler(fd, pSIdTagToken)
    int             fd;
    shastraIdTag   *pSIdTagToken;
{
    putStringOnChannel(fd, REQ_ASK_TOKEN, "putCollAskTokenHandler()");
    ShastraIdTagOut(fd, pSIdTagToken);
    cmFlush(fd);
}




/*
 * Function
 */
closedChannelCleanUpHandler(fd)
    int             fd;
{
    if (shaKernFlags[fd] == SHAFRONT) {
        collLeaveCleanUpHandler(fd);
    } else {
        mplexUnRegisterChannel(fd);
/* CHECK actually initiate retry-connection sequence */
    }
}

/*
 * Function
 */
int putCollCommMsgTextHandler(fd, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
    int fd;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COMM_MSGTEXT, "putCollCommMsgTextHandler()")
        ;
    ShastraIdTagOut(fd, pSmSIdTag);
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
}

/*
 * Function
 */
int collCommMsgTextHandler(fd)
    int fd;
{
    shastraIdTag    smSIdTag;
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collCommMsgTextHandler()")){
        case route_FRONT:
            putCollCommMsgTextHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGTEXT);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
```

```
int putCollCommMsgTextFileHandler(fd, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
    int fd;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COMM_MSGTEXTFILE,
        "putCollCommMsgTextFileHandler()");
    ShastraIdTagOut(fd, pSmSIdTag);
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
}

/*
 * Function
 */
int collCommMsgTextFileHandler(fd)
    int fd;
{
    shastraIdTag    smSIdTag;
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collCommMsgTextFileHandler()")){
        case route_FRONT:
            putCollCommMsgTextFileHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGTEXTFILE);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int putCollCommMsgAudioHandler(fd, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
```

```
    int fd;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COMM_MSGAUDIO, "putCollCommMsgAudioHandler(
        )");
    ShastraIdTagOut(fd, pSmSIdTag);
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
}

/*
 * Function
 */
int collCommMsgAudioHandler(fd)
    int fd;
{
    shastraIdTag    smSIdTag;
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collCommMsgAudioHandler()")){
        case route_FRONT:
            putCollCommMsgAudioHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGAUDIO);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int putCollCommMsgAudioFileHandler(fd, pSIdTag, pToSIdTag, pSmSIdTag, sbMsg
    )
    int fd;
    shastraIdTag *pSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSmSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COMM_MSGAUDIOFILE,
        "putCollCommMsgAudioFileHandler()");
    ShastraIdTagOut(fd, pSmSIdTag);
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
}

/*
 * Function
 */
int collCommMsgAudioFileHandler(fd)
    int fd;
{
    shastraIdTag    smSIdTag;
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collCommMsgAudioFileHandler()")){
        case route_FRONT:
            putCollCommMsgAudioFileHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGAUDIOFILE);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int putCollCommMsgVideoHandler(fd, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
    int fd;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COMM_MSGVIDEO, "putCollCommMsgVideoHandler(
        )");
    ShastraIdTagOut(fd, pSmSIdTag);
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
}

/*
 * Function
 */
int collCommMsgVideoHandler(fd)
    int fd;
{
    shastraIdTag    smSIdTag;
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collCommMsgVideoHandler()")){
        case route_FRONT:
            putCollCommMsgVideoHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGVIDEO);
    showInfo(sbOutMsgBuf);
}

/*
 * Function
 */
int putCollCommMsgVideoFileHandler(fd, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg
    )
    int fd;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COMM_MSGVIDEOFILE,
        "putCollCommMsgVideoFileHandler()");
    ShastraIdTagOut(fd, pSmSIdTag);
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
}

/*
 * Function
 */
int collCommMsgVideoFileHandler(fd)
    int fd;
{
    shastraIdTag    smSIdTag;
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collCommMsgVideoFileHandler()")){
        case route_FRONT:
            putCollCommMsgVideoFileHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGVIDEOFILE);
    showInfo(sbOutMsgBuf);
}
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
#include <stdio.h>

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>

#include <Xm/Form.h>
#include <Xm/Label.h>
#include <Xm/Text.h>
#include <Xm/RowColumn.h>

#include <shastra/uitools/strListUtilities.h>
#include <shastra/uitools/buttonBox.h>
#include <shastra/uitools/confirmCB.h>
#include <shastra/uitools/chooseOne.h>
#include <shastra/uitools/callbackArg.h>

#include <shastra/datacomm/shastraIdH.h>
#include <shastra/datacomm/shastraIdTagH.h>

#include <shastra/shautils/shautils.h>
#include <shastra/shautils/kernelFronts.h>
#include <shastra/shautils/sesMgrFronts.h>

#include <shastra/session/sesMgrMainCB.h>
#include <shastra/session/sesMgr.h>
#include <shastra/session/sesMgr_client.h>
#include <shastra/session/sesMgrState.h>
```

```
/*
 * Function: createMainCmdShell (private)
 *
 */

Widget
createMainCmdShell(wgParent)
    Widget          wgParent;
{
    Widget          wgMainCmdShell, wgMainCmdForm;
    Widget          wgName;
    XmString        xmName;
    char *sName;

    /* Create the menu popup shell */
    wgMainCmdShell = XtVaCreatePopupShell("mainCmdShell",
                  topLevelShellWidgetClass, wgParent, NULL);

    /*
     * Create the menu form widget used to position the widgets inside
     * the
     */
    /* menu window */
    wgMainCmdForm = XtVaCreateManagedWidget("mainCmdForm",
        xmFormWidgetClass,
                        wgMainCmdShell, NULL);

    sName = resolveNameFrom2Bases(pSesMgrAppData->sDirBase,
                    pSesMgrAppData->sDirDefs, "bitmaps/terminal.xbm");
    wgName = XtVaCreateManagedWidget("hostNameLabel", xmLabelWidgetClass,
                     wgMainCmdForm,
                     XmNbackgroundPixmap,
                        convertStringToPixmap(wgMainCmdForm, sName),
                     NULL);
    xmName = XmStringCreateSimple(shortenName(kernelHostName));
    XtVaSetValues(wgName, XmNlabelString, (XtArgVal) xmName, NULL);
    XmStringFree(xmName);

    /*
     * Create the button box and state box objects that are inside the
     * menu
     */
    /* window */

    createMainCmdButtonBox(wgMainCmdForm);
    createMainDbgButtonBox(wgMainCmdForm);
    createTextStatusBox(wgMainCmdForm);

    return wgMainCmdShell;
}

/*
 * Function: createMainCmdButtonBox (private)
 */
Widget           wgMainKill;
Widget           wgMainQuit;
chooseOne       *pcoShastraSesMgr;
chooseOne       *pcoShastraKern;
chooseOne       *pcoShastraFront;
chooseOne       *pcoShastraSys;
char           **rgsbShastraKern;
char           **rgsbShastraSesMgr;
char           **rgsbShastraFront;
char           **rgsbShastraSys;
char            *rgsbNull[] = {NULL};

void
createMainCmdButtonBox(wgParent)
    Widget          wgParent;
{
    static button   abu[] = {
        {"kill", &wgMainKill},
        {"quit", &wgMainQuit},
        {NULL, NULL}
    };

    buttonBoxCreate("mainBtnsBox", wgParent, abu, True);

    /* Create a choose one object to select one system */
    pcoShastraFront = chooseOneCreate(NULL, coNoInitialHighlight,
                      wgMainKill, chooseOneTestCB,
                    (XtPointer) pcbArgPopup, wgMainKill,
                       "Choose Local Front-end", 200, NULL);
    chooseOneChangeList(pcoShastraFront, rgsbNull, coNoInitialHighlight);

    /* Create a choose one object to select one system */
    pcoShastraSesMgr = chooseOneCreate(NULL, coNoInitialHighlight,
                       wgMainKill, chooseOneTestCB,
                    (XtPointer) pcbArgPopup, wgMainKill,
                     "Choose Remote SesMgr", 200, NULL);
    chooseOneChangeList(pcoShastraSesMgr, rgsbNull, coNoInitialHighlight);

    /* Create a choose one object to select one system */
    pcoShastraKern = chooseOneCreate(NULL, coNoInitialHighlight,
                     wgMainKill, chooseOneTestCB,
                     (XtPointer) pcbArgPopup, wgMainKill,
                     "Choose Remote Kernel", 200, NULL);
    chooseOneChangeList(pcoShastraKern, rgsbNull, coNoInitialHighlight);

    /* Create a choose one object to select one system */
    pcoShastraSys = chooseOneCreate(NULL, coNoInitialHighlight,
                    wgMainKill, chooseOneTestCB,
                    (XtPointer) pcbArgPopup, wgMainKill,
                    "Choose Remote System", 200, NULL);
    chooseOneChangeList(pcoShastraSys, rgsbNull, coNoInitialHighlight);
    XtAddCallback(wgMainQuit, XmNactivateCallback, mainQuitCB, NULL);
    XtAddCallback(wgMainKill, XmNactivateCallback, mainKillCB,
              (XtPointer) pcoShastraFront);
}

/*
 * Function: createTextStatusBox (private)
 */
Widget          wgStatusText;

void
createTextStatusBox(wgParent)
    Widget          wgParent;
{
    Arg             args[8];
    int             n;

    n = 0;
    XtSetArg(args[n], XmNrows, 5);
    n++;
    XtSetArg(args[n], XmNcolumns, 40);
    n++;
    XtSetArg(args[n], XmNeditable, False);
    n++;
    XtSetArg(args[n], XmNeditMode, XmMULTI_LINE_EDIT);
    n++;
    XtSetArg(args[n], XmNscrollBarDisplayPolicy, XmAS_NEEDED);
    n++;
    wgStatusText = XmCreateScrolledText(wgParent, "mainStatusText",
                        args, n);
    XtManageChild(wgStatusText);
}


/*
 * Function: createMainDbgButtonBox (private)
 */
Widget          wgDbgCheckSys;
Widget          wgDbgGetSys;
Widget          wgDbgGetKern;
Widget          wgDbgCheckSmFr;
Widget          wgDbgGetSmFr;
Widget          wgDbgGetSesm;

void
createMainDbgButtonBox(wgParent)
    Widget          wgParent;
{
    static button   abu[] = {
        {"getKern", &wgDbgGetKern},
        {"getSys", &wgDbgGetSys},
        {"checkSys", &wgDbgCheckSys},
        {"getSesm", &wgDbgGetSesm},
        {"getSmFr", &wgDbgGetSmFr},
        {"checkSmFr", &wgDbgCheckSmFr},
        {NULL, NULL}
    };

    buttonBoxCreate("dbgBtnsBox", wgParent, abu, True);

    XtAddCallback(wgDbgCheckSys, XmNactivateCallback, dbgCheckSysCB,
              (XtPointer) pcoShastraKern);
    XtAddCallback(wgDbgGetSys, XmNactivateCallback, dbgGetSysCB,
              (XtPointer) pcoShastraKern);
    XtAddCallback(wgDbgGetKern, XmNactivateCallback, dbgGetKernCB,
              (XtPointer) NULL);
    XtAddCallback(wgDbgCheckSmFr, XmNactivateCallback, dbgCheckSmFrCB,
              (XtPointer) pcoShastraSesMgr);
    XtAddCallback(wgDbgGetSmFr, XmNactivateCallback, dbgGetSmFrCB,
              (XtPointer) pcoShastraSesMgr);
    XtAddCallback(wgDbgGetSesm, XmNactivateCallback, dbgGetSesmCB,
              (XtPointer) NULL);
}

void
mainKillCB(widget, xpClientData, call_data)
    Widget          widget;
    XtPointer       xpClientData, call_data;
{
    chooseOne      *pco = (chooseOne *) xpClientData;

    strcpy(pcbArgPopup->msg, "chooseSystem");
    pcbArgPopup->operation = endSystemOprn;
    pcbArgPopup->fWantOprn = 1;
    pcbArgPopup->fWantArg = 0;  /* no call for name */
    pcbArgPopup->wgInitiator = widget;

    /* Pop up the choose one object */
    chooseOneMobExec(pco, widget);

}

void
mainQuitCB(widget, closure, call_data)
    Widget          widget;
    XtPointer       closure, call_data;
{
    strcpy(pcbArgPopup->msg, "Confirm Action");
    strcpy(pcbArgPopup->prompt, "Please Confirm Action");
    pcbArgPopup->operation = quitOprn;
    pcbArgPopup->fWantOprn = 1;
    pcbArgPopup->fWantArg = 0;  /* call for name */
    pcbArgPopup->wgInitiator = widget;
    ConfirmPopup(widget);

}
```

```
void
dbgCheckSysCB(wg, xpClientData, call_data)
    Widget          wg;
    XtPointer       xpClientData, call_data;
{
    chooseOne      *pco = (chooseOne *) xpClientData;

    strcpy(pcbArgPopup->msg, "chooseKernel");
    pcbArgPopup->operation = dbgCheckSysOprn;
    pcbArgPopup->fWantOprn = 1;
    pcbArgPopup->fWantArg = 0;  /* no call for name */
    pcbArgPopup->wgInitiator = wg;

    /* Pop up the choose one object */
    chooseOneMobExec(pco, wg);

}


void
dbgGetSysCB(wg, xpClientData, call_data)
    Widget          wg;
    XtPointer       xpClientData, call_data;
{
    chooseOne      *pco = (chooseOne *) xpClientData;

    strcpy(pcbArgPopup->msg, "chooseKern");
    pcbArgPopup->operation = getShaKernFrIdOprn;
    pcbArgPopup->fWantOprn = 1;
    pcbArgPopup->fWantArg = 0;  /* no call for name */
    pcbArgPopup->wgInitiator = wg;

    /* Pop up the choose one object */
    chooseOneMobExec(pco, wg);

}

void
dbgGetKernCB(wg, xpClientData, call_data)
    Widget          wg;
    XtPointer       xpClientData, call_data;
{
    getShaKernIdOprn(0);
}

void
dbgCheckSysOprn(iObjIndex)
    int             iObjIndex;
{
    shastraIds     *pSIds;
    shastraId      *pSId;
    int             kernFd;

    pSId = shastraKernIds.shastraIds_val[iObjIndex];
    kernFd = locateKernFronts(pSId);
    if (kernFd < 0) {
        fprintf(stderr, "dbgCheckSysOprn()->kernFd = %d\n", kernFd);
        return;
    }
    pSIds = getKernFrontSIds(pSId);

    if (rgsbShastraSys != NULL) {
        strListDestroy(rgsbShastraSys);
    }
    rgsbShastraSys = pSIds2StrTab(pSIds, PSIDSHOWALL);
    chooseOneChangeList(pcoShastraSys, rgsbShastraSys, coNoInitialHighlight
        );

    strcpy(pcbArgPopup->msg, "chooseSys");
    pcbArgPopup->operation = NULL;
    pcbArgPopup->fWantOprn = 0;
    pcbArgPopup->fWantArg = 0;  /* no call for name */

    /* Pop up the choose one object */
    chooseOneMobExec(pcoShastraSys, pcbArgPopup->wgInitiator);
}




void
dbgCheckSmFrCB(wg, xpClientData, call_data)
    Widget          wg;
    XtPointer       xpClientData, call_data;
{
    chooseOne      *pco = (chooseOne *) xpClientData;

    strcpy(pcbArgPopup->msg, "chooseSesMgr");
    pcbArgPopup->operation = dbgCheckSmFrOprn;
    pcbArgPopup->fWantOprn = 1;
    pcbArgPopup->fWantArg = 0;  /* no call for name */
    pcbArgPopup->wgInitiator = wg;

    /* Pop up the choose one object */
    chooseOneMobExec(pco, wg);

}


void
dbgGetSmFrCB(wg, xpClientData, call_data)
    Widget          wg;
    XtPointer       xpClientData, call_data;
{
    chooseOne      *pco = (chooseOne *) xpClientData;

    strcpy(pcbArgPopup->msg, "chooseSesm");
    pcbArgPopup->operation = getShaSesmFrIdOprn;
    pcbArgPopup->fWantOprn = 1;
    pcbArgPopup->fWantArg = 0;  /* no call for name */
    pcbArgPopup->wgInitiator = wg;

    /* Pop up the choose one object */
    chooseOneMobExec(pco, wg);

}

void
dbgGetSesmCB(wg, xpClientData, call_data)
    Widget          wg;
    XtPointer       xpClientData, call_data;
{
    getShaSesmIdOprn(0);
}

void
dbgCheckSmFrOprn(iObjIndex)
    int             iObjIndex;
{
    shastraIdTags  *pSIdTags;
    shastraIdTag   *pSIdTag;
    int             smIndex;

    pSIdTag = (shastraIdTag *) & shastraSesmIds.shastraIds_val[iObjIndex]->
        lSIDTag;
    smIndex = locateSesmFronts(pSIdTag);
    if (smIndex < 0) {
        fprintf(stderr, "dbgCheckSysOprn()->smIndex = %d\n", smIndex);
        return;
    }
    pSIdTags = getSesmFrontSIdTags(pSIdTag);

    if (rgsbShastraSys != NULL) {
        strListDestroy(rgsbShastraSys);
    }
    rgsbShastraSys = mapSIdTags2StrTab(pSIdTags, PSIDSHOWALL);
    chooseOneChangeList(pcoShastraSys, rgsbShastraSys, coNoInitialHighlight
        );

    strcpy(pcbArgPopup->msg, "chooseSys");
    strcpy(pcbArgPopup->prompt, "Enter Password:");
    pcbArgPopup->operation = endSystemOprn;
    pcbArgPopup->fWantOprn = 1;
    pcbArgPopup->fWantArg = 1;  /* call for name */

    /* Pop up the choose one object */
    chooseOneMobExec(pcoShastraSys, pcbArgPopup->wgInitiator);
```

```
}


/*
 * Function --
 */
void
outputTextToWidget(s, wg, pCurrentPosn)
    char               *s;
    Widget              wg;
    XmTextPosition *pCurrentPosn;
{
    XmTextBlock     textBlock;
    XmTextPosition  currentPosn;

    if (pCurrentPosn == 0) {
        currentPosn = XmTextGetInsertionPosition(wg);
        pCurrentPosn = &currentPosn;
    } else {
        XmTextSetInsertionPosition(wg, *pCurrentPosn);
    }
    XmTextReplace(wg, *pCurrentPosn, *pCurrentPosn, s);
    *pCurrentPosn += strlen(s);

#ifdef WANTTHIS
    /* Save output in buffer */
    if (strlen(saveBuffer) + strlen(s) + 1 <= MAXLEN) {
        strcat(saveBuffer, s);
    } else {
        printf("Save-buffer overflow.\n");
    }
#endif              /* WANTTHIS */
}
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
#include <stdio.h>

#include <shastra/draw/drawdata.h>
#include <shastra/draw/pict.h>
#include <shastra/network/mplex.h>
#include <shastra/network/server.h>
#include <shastra/solid/imageIO.h>

void            generateContoursFromPict(Prot5(pictData *, int, int, int,
    int));

mLineData      *
readLineImageFD(fd)
    int             fd;
{
    int             i, j;

    mLineData      *mLine;
    lineData       *line;
    char *sbIn;

    mLine = (mLineData *) malloc(sizeof(mLineData));

    sbIn = cmReceiveString(fd);
    sscanf(sbIn, "%d", &mLine->nLines);
    free(sbIn);

    mLine->lines = (lineData *) malloc(sizeof(lineData) *
                        mLine->nLines);
    for (i = 0; i < mLine->nLines; i++) {
        line = &mLine->lines[i];
        sbIn = cmReceiveString(fd);
        sscanf(sbIn, "%d", &line->number);
        free(sbIn);
        line->array = (double (*)[3]) malloc(sizeof(double) *
                             3 * line->number);

        for (j = 0; j < line->number; j++) {
            sbIn = cmReceiveString(fd);
            sscanf(sbIn, "%lf%lf%lf",
                   &line->array[j][0],
                   &line->array[j][1],
                   &line->array[j][2]);
            free(sbIn);
        }
    }
    return mLine;
}
mLineData      *
readLineImage(inStream)
    FILE           *inStream;
{
    int             i, j;

    mLineData      *mLine;
    lineData       *line;

    mLine = (mLineData *) malloc(sizeof(mLineData));
    fscanf(inStream, "%d", &mLine->nLines);
    mLine->lines = (lineData *) malloc(sizeof(lineData) *
                       mLine->nLines);
    for (i = 0; i < mLine->nLines; i++) {
        line = &mLine->lines[i];
        fscanf(inStream, "%d", &line->number);
        line->array = (double (*)[3]) malloc(sizeof(double) *
                             3 * line->number);

        for (j = 0; j < line->number; j++) {
            fscanf(inStream, "%lf%lf%lf",
                   &line->array[j][0],
                   &line->array[j][1],
                   &line->array[j][2]);
        }
    }
    return mLine;
}

void
writeLineImageFD(fd, mLine)
    int             fd;
    mLineData      *mLine;
{
    int             i, j;
    lineData       *line;
    char sbOut[256];

    sprintf(sbOut, "%d\n", mLine->nLines);
    cmSendString(fd,sbOut);
    for (i = 0; i < mLine->nLines; i++) {
        line = &mLine->lines[i];
        sprintf(sbOut, "%d\n", line->number);
        cmSendString(fd,sbOut);

        for (j = 0; j < line->number; j++) {
            sprintf(sbOut, "%lf %lf %lf\n",
                line->array[j][0],
                line->array[j][1],
                line->array[j][2]);
            cmSendString(fd,sbOut);
        }
    }
}
void
writeLineImage(outStream, mLine)
    FILE           *outStream;
    mLineData      *mLine;
{
    int             i, j;
    lineData       *line;

    fprintf(outStream, "%d\n", mLine->nLines);
    for (i = 0; i < mLine->nLines; i++) {
        line = &mLine->lines[i];
        fprintf(outStream, "%d\n", line->number);

        for (j = 0; j < line->number; j++) {
            fprintf(outStream, "%lf %lf %lf\n",
                line->array[j][0],
                line->array[j][1],
                line->array[j][2]);
        }
    }
}

void
freeLineImage(mLine)
    mLineData      *mLine;
{
    int             i, j;
    lineData       *line;

    for (i = 0; i < mLine->nLines; i++) {
        line = &mLine->lines[i];
        free(line->array);
    }
    free(mLine->lines);
    free(mLine);
}


mLineData      *
copyLineImage(inmLine)
    mLineData      *inmLine;
{
    int             i, j;

    mLineData      *mLine;
    lineData       *line;
    lineData       *inLine;

    mLine = (mLineData *) malloc(sizeof(mLineData));
    mLine->nLines = inmLine->nLines;
    mLine->lines = (lineData *) malloc(sizeof(lineData) *
                        mLine->nLines);
    for (i = 0; i < mLine->nLines; i++) {
        line = &mLine->lines[i];
        inLine = &inmLine->lines[i];
        line->number = inLine->number;
        line->array = (double (*)[3]) malloc(sizeof(double) *
                              3 * line->number);

        memcpy(line->array, inLine->array, sizeof(double) *
              3 * line->number);
    }
    return mLine;
}

int
sendPictContours(fd, pPict)
    int             fd;
    pictData       *pPict;
{
    mLineData       mLine;

    generateContoursFromPict(pPict, 1/*fBern*/, 1/*fCircEll*/,
        24/*iPieces*/, 0/*iForLamina*/);
    mLine.nLines = pPict->nPicts;
    mLine.lines = pPict->contours;
    writeLineImageFD(fd, &mLine);
    return 1;
}
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/*
 * convert.c
 */

#include <stdio.h>

#include <poly/poly.h>
#include <poly/polymath.h>
#include <shastra/solid/datadefs.h>
#include <shastra/solid/edgetypes.h>
#include <shastra/solid/eqntypes.h>
#include <shastra/solid/macros.h>
#include <shastra/solid/readSolid.h>
#include <ipoly/iPolyH.h>
#include <ipoly/ipolyutil.h>

#define DEBUG 0
#define iabs(x) ((x) < 0 ? -(x) : (x))

extern char *stdVars[3];



Solid_Ptr
convertIPolyToSolid(pIPoly)
     iPoly *pIPoly;
{
  Stack_Union solObject;
  int i,j;
```

```
    int nDEs = 0;
    Solid_Ptr pSolid = createSolid();
    Vertex_Ptr pVertex;
    Edge_Ptr pEdge;
    Face_Ptr pFace;
    Cycle_Ptr pCycle;
    DEdge_Ptr pDEdge;

    strcpy(pSolid->name, "iPolySolid");
    if (DEBUG) {
     fprintf(stdout, "######solid#######\n\n");
     fprintf(stdout, "SOLID %s\n", pSolid->name);
     fprintf(stdout, "%d %d %d\t#vertices, edges, faces\n",
       IPolyNVerts(pIPoly), IPolyNEdges(pIPoly), IPolyNFaces(pIPoly));
    }
    for (i = 0; i < IPolyNVerts(pIPoly); i++) {
     pVertex = createVertex();
     solObject.vertex = pVertex;
     AddObjToSolid(&solObject, VERTEX, pSolid);
    }
    for (i = 0; i < IPolyNEdges(pIPoly); i++) {
     pEdge = createEdge();
     solObject.edge = pEdge;
     AddObjToSolid(&solObject, EDGE, pSolid);
    }
   /*CHECK -- assuming #faces == #cycles.. true except for grouped objects..*/
    for (i = 0; i < IPolyNFaces(pIPoly); i++) {
     pFace = createFace();
     solObject.face = pFace;
     AddObjToSolid(&solObject, FACE, pSolid);
     pCycle = createCycle();
     solObject.cycle = pCycle;
     AddObjToSolid(&solObject, CYCLE, pSolid);
    }

   /*  if((IPolyNVertFaceAdjs(pIPoly) == 0) || (IPolyNVertEdgeAdjs(pIPoly) ==
     0)){
     genIPolyAdjInfo(pIPoly);
    }*/

    if (DEBUG) {
     fprintf(stdout, "######vertices#######\n\n");
    }
    for (i = 0; i < IPolyNVerts(pIPoly); i++) {
     Vertex_Ptr pVertex = Solid_Vertex(pSolid, i);
     double *point;
     int iV = i+1;

     point = IPolyVert(pIPoly, i);
     sprintf(pVertex->name, "v%d", iV);
     pVertex->point[0] = point[0];
     pVertex->point[1] = point[1];
     pVertex->point[2] = point[2];
    if (DEBUG) {
      fprintf(stdout, "%lf %lf %lf\t#point for v%d\n",
          point[0], point[1], point[2], iV);
    }

    if((IPolyNVertFaceAdjs(pIPoly) > 0) &&
       (IPolyNVertEdgeAdjs(pIPoly) > 0)){
/*have vert face and edge adjs, use to compute adj info*/
      for (j = 0; j < IPolyVertNFaceAdjs(pIPoly, i); j++) {
    IPolyVertFaceAdj(pIPoly, i, j);
      }
      for (j = 0; j < IPolyVertNEdgeAdjs(pIPoly, i); j++) {
    IPolyVertEdgeAdj(pIPoly, i, j);
      }
    }
/*    fDoneVertAdjs; */
  }

  if (DEBUG) {
    fprintf(stdout, "#####edges#######\n");
  }
  for (i = 0; i < IPolyNEdges(pIPoly); i++) {
    Edge_Ptr pEdge = Solid_Edge(pSolid, i);
    Vertex_Ptr v1, v2;
    int iE = i+1;
    int iV1, iV2;

    sprintf(pEdge->name, "e%d", iE);
    iV1 = IPolyEdgeV1(pIPoly, i) +1;
    iV2 = IPolyEdgeV2(pIPoly, i) +1;
    fillIndex(&pEdge->vertex1,0,VERTEX,iV1);
    fillIndex(&pEdge->vertex2,0,VERTEX,iV2);
    if (DEBUG) {
      fprintf(stdout, "%s\t#name for e%d\n", pEdge->name, iE);
      fprintf(stdout, "V %d\t#vert1 for e%d\n", iV1, iE);
      fprintf(stdout, "V %d\t#vert2 for e%d\n", iV2, iE);
    }
    pEdge->type = LINEAR;
    v1 = Solid_Vertex(pSolid, iV1 - 1);
    v2 = Solid_Vertex(pSolid, iV2 - 1);
    pEdge->tan12[0] = v2->point[0] - v1->point[0];
    pEdge->tan12[1] = v2->point[1] - v1->point[1];
    pEdge->tan12[2] = v2->point[2] - v1->point[2];
    normalizeDblVector(pEdge->tan12);
    pEdge->tan21[0] = -pEdge->tan12[0];
    pEdge->tan21[1] = -pEdge->tan12[1];
    pEdge->tan21[2] = -pEdge->tan12[2];
    if (DEBUG) {
      fprintf(stdout, "%lf %lf %lf\t#tan12 for e%d\n",
          pEdge->tan12[0], pEdge->tan12[1], pEdge->tan12[2], iE);
      fprintf(stdout, "%lf %lf %lf\t#tan21 for e%d\n",
          pEdge->tan21[0], pEdge->tan21[1], pEdge->tan21[2], iE);
    }
    if(IPolyNEdgeFaceAdjs(pIPoly) > 0){
/*have edge face adjs, use to get dedge info*/
      for (j = 0; j < IPolyEdgeNFaceAdjs(pIPoly, i); j++) {
    IPolyEdgeFaceAdj(pIPoly, i, j);
      }
    }
/* fDoneEdgeDEs = 1*/
  }

  if (DEBUG) {
    fprintf(stdout, "#####faces#######\n");
  }
  for (i = 0; i < IPolyNFaces(pIPoly); i++) {
    CycleList_Ptr pCycPtr;
    DEList_Ptr pDEPtr;
    AdjList_Ptr pAdjPtr;
    int iF = i+1;
    int iD, iND, iPD, iV;
    Poly PlaneEqnFrom3Pts();

    pFace = Solid_Face(pSolid, i);
    sprintf(pFace->name, "f%d", iF);
    pFace->type = IMPLICIT;
    if (DEBUG) {
      fprintf(stdout, "%s\t#name for f%d\n", pFace->name, iF);
    }
    if(IPolyNFaceVerts(pIPoly, i) >= 3){
      pFace->equation =
    PlaneEqnFrom3Pts(IPolyVert(pIPoly, IPolyFaceVert(pIPoly, i, 0)),
             IPolyVert(pIPoly, IPolyFaceVert(pIPoly, i, 1)),
             IPolyVert(pIPoly, IPolyFaceVert(pIPoly, i, 2)));
    }
    else{
      pFace->equation = Parse("x + y + z");
    }
    ConformPolyToVars(3, stdVars, pFace->equation);

    pFace->normal = createEqnItem();
    pFace->normal->eQN = DiffPoly(pFace->equation, 0);
    ConformPolyToVars(3, stdVars, pFace->normal->eQN);
    pFace->normal->next = createEqnItem();
    pFace->normal->next->eQN = DiffPoly(pFace->equation, 1);
    ConformPolyToVars(3, stdVars, pFace->normal->next->eQN);
    pFace->normal->next->next = createEqnItem();
    pFace->normal->next->next->eQN = DiffPoly(pFace->equation, 2);
    ConformPolyToVars(3, stdVars, pFace->normal->next->next->eQN);
    if (DEBUG) {
      fprintf(stdout, "%s\t#Equation for f%d\n",
          UnParse(pFace->equation), iF);
      fprintf(stdout, "%s\t#X normal component for f%d\n",
          UnParse(pFace->normal->eQN), iF);
      fprintf(stdout, "%s\t#Y normal component for f%d\n",
          UnParse(pFace->normal->next->eQN), iF);
```

```
        fprintf(stdout, "%s\t#Z normal component for f%d\n",
            UnParse(pFace->normal->next->next->eQN), iF);
      }
      if (DEBUG) {
        fprintf(stdout, "1\t#number of cycles for f%d\n", iF);
      }
      pCycPtr = createCycleItem();
      pCycPtr->next = pFace->cycles;
      pFace->cycles = pCycPtr;
      if (DEBUG) {
        fprintf(stdout, "C %d\t#cycle for f%d\n", iF, iF);
      }
      fillIndex(&pCycPtr->cycle,0,CYCLE,iF);

      pCycle = Solid_Cycle(pSolid, i);
      if (DEBUG) {
        fprintf(stdout, "F %d\t#face for c%d\n", iF, iF);
      }
      fillIndex(&pCycle->face,0,FACE,iF);

      if((IPolyNEdgeFaces(pIPoly) > 0) &&
         (IPolyNEdgeFaces(pIPoly) == IPolyNVertFaces(pIPoly))){
  /*have faces by edge and vertex, use to compute dedges, adj info*/
        for (j = 0; j < IPolyNFaceEdges(pIPoly, i); j++) {
      pDEdge = createDEdge();
      solObject.dEdge = pDEdge;
      AddObjToSolid(&solObject, DEDGE, pSolid);
      nDEs ++;

      iD = IPolyFaceEdge(pIPoly, i, j);
      iND = (j==IPolyNFaceEdges(pIPoly, i)-1)?
        nDEs-IPolyNFaceEdges(pIPoly, i)+1: nDEs+1;
      iPD = (j==0)?
        nDEs+IPolyNFaceEdges(pIPoly, i)-1: nDEs-1;
      pEdge = Solid_Edge(pSolid, iabs(iD)-1);

      pDEPtr = createDEdgeItem();
      pDEPtr->next = pEdge->dEdges;
      pEdge->dEdges = pDEPtr;
      if (DEBUG) {
        fprintf(stdout, "D %d\t#dedge for e%d\n", nDEs, iabs(iD));
      }
      fillIndex(&pDEPtr->dEdge,0,DEDGE, nDEs);

      if (DEBUG) {
        fprintf(stdout, "E %d\t#edge for de%d\n", iabs(iD), nDEs);
        fprintf(stdout, "C %d\t#cycle for de%d\n", iF, nDEs);
        fprintf(stdout, "RO %d\t#orientn for de%d\n", iD>0?1:0, nDEs);
        fprintf(stdout, "D %d\t#nextde for de%d\n", iND, nDEs);
      }
      pDEdge->rightOrientation = (iD>0)?1:0;
      fillIndex(&pDEdge->edge,0,EDGE,iabs(iD));
      fillIndex(&pDEdge->cycle,0,FACE,iF);
```

```
        fillIndex(&pDEdge->nextDE,0,DEDGE,iND);
        if(j==0){
          if (DEBUG) {
            fprintf(stdout, "D %d\t#dedge for c%d\n", nDEs, iF);
          }
          fillIndex(&pCycle->dEdge,0,DEDGE,nDEs);
        }

        iV = IPolyFaceVert(pIPoly, i, j)+1;/*indexed from 0*/
        pVertex = Solid_Vertex(pSolid, iV-1);
        pAdjPtr = createAdjItem();
        pAdjPtr->next = pVertex->adjacencies;
        pVertex->adjacencies = pAdjPtr;
        fillIndex(&pAdjPtr->face, 0, FACE, iF);
        fillIndex(&pAdjPtr->dEIn, 0, DEDGE, iPD);
        fillIndex(&pAdjPtr->dEOut, 0, DEDGE, nDEs);
        if (DEBUG) {
          fprintf(stdout, "F %d\t#face adj for v%d\n", iF, iV);
          fprintf(stdout, "D %d\t#dedge in for v%d\n",
              pAdjPtr->dEIn.index, iV);
          fprintf(stdout, "D %d\t#dedge out for v%d\n",
              pAdjPtr->dEOut.index, iV);
        }
          }
        }
        else{
          fprintf(stderr,"convertIPolyToSolid()->inconsistency in iPoly!\n");
        }
      }
    /*
      if(!fDoneVertAdjs){
        setAllVertexAdjacencies(pSolid);
      }
    */

      return pSolid;
    }
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/*********************************************************************/
/*
 * copySolid.c - input functions for solid at the network interface
 *
 * copyString()
 *
 * copyIndex() copyAdjItem() copyEqnItem()
 *
 * copyVertex() copyDEdge() copyEdge() copyCycle() copyFace() copySolid()
 *
 */
/********************************************************************/

#include <stdio.h>
#include <ctype.h>

#include <shastra/shilp.h>
#include <shastra/solid/datadefs.h>
#include <shastra/solid/macros.h>
#include <shastra/solid/bern.h>

#include <poly/poly.h>
#include <poly/polymath.h>
#include <shastra/solid/readSolid.h>
#include <shastra/solid/copySolid.h>

/*
 * copyIndex(inIndex, iptr) - copy an index
 *
 */
void
copyIndex(inIndex, iptr)
    Index_Ptr       inIndex, iptr;
{
    memcpy(iptr, inIndex, sizeof(Index_Struct));
}

/********************************************************************/
/*
 * copyAdjItem( inAdjItem,aptr ) - copy an adjacency into item pointer
 *
 */
/********************************************************************/
void
copyAdjItem(inAdjItem, aptr)
    AdjList_Ptr     inAdjItem, aptr;
{
    copyIndex(&inAdjItem->face, &aptr->face);
    copyIndex(&inAdjItem->dEIn, &aptr->dEIn);
    copyIndex(&inAdjItem->dEOut, &aptr->dEOut);
}

/********************************************************************/
/*
 * copyEqnItem(inEqnItem ) - copy an equation item, create it and return it
 */
/********************************************************************/
EQNList_Ptr
copyEqnItem(inEqnItem)
    EQNList_Ptr     inEqnItem;
{
    EQNList_Ptr     New_Eqn = createEqnItem();

    New_Eqn->eQN = CopyPoly(inEqnItem->eQN);

    return (New_Eqn);
}

/********************************************************************/
/*
 * reverseBernPar( inEqn) - reverse bernstein-parametric eqn
 *
 */
/********************************************************************/
void
reverseBernPar(inEqn)
    BernPar_Ptr     inEqn;
{
    int             i;
    int             n, n2;
    double          tmpBuf[3];
    if ((inEqn == NULL) || (inEqn->degree == 0)) {
        return;
    }
    n = (1 + inEqn->degree);
    n2 = n / 2;
    for (i = 0; i < n2; i++) {
        memcpy(tmpBuf, inEqn->coeffs[i], 3 * sizeof(double));
        memcpy(inEqn->coeffs[i], inEqn->coeffs[n - i], 3 * sizeof(double));
        memcpy(inEqn->coeffs[n - i], tmpBuf, 3 * sizeof(double));
    }
    return;
}
/********************************************************************/
/*
 * copyBernPar( inEqn) - copy bernstein-parametric eqn, return pointer
 *
 */
/********************************************************************/
BernPar_Ptr
copyBernPar(inEqn)
    BernPar_Ptr     inEqn;
{
    int             i;
    BernPar_Ptr     eqn;
    if (inEqn == NULL) {
        return NULL;
    }
    eqn = (BernPar_Ptr) malloc(sizeof(BernPar));
    eqn->degree = inEqn->degree;
    if (eqn->degree > 0) {
        eqn->coeffs = (double (*)[3])
            createMem(3 * (1 + eqn->degree) * sizeof(double));
        memcpy(eqn->coeffs,inEqn->coeffs,
              3 * (1 + eqn->degree) * sizeof(double));
    }
    return eqn;
}
/********************************************************************/
/*
 * reverseBernParQuad( inEqn) - reverse bernstein-parametric quad eqn
 *
 */
/********************************************************************/
void
reverseBernParQuad(inEqn)
    BernParQuad_Ptr inEqn;
{
    int             i;
    int             n, n2;
    double          tmpBuf[3];
    if ((inEqn == NULL) || (inEqn->degree == 0)) {
        return;
    }
    n = (1 + inEqn->degree);
    n2 = n / 2;
    for (i = 0; i < n2; i++) {
        memcpy(tmpBuf, inEqn->coeff1[i], 3 * sizeof(double));
        memcpy(inEqn->coeff1[i], inEqn->coeff1[n - i], 3 * sizeof(double));
        memcpy(inEqn->coeff1[n - i], tmpBuf, 3 * sizeof(double));
    }
    for (i = 0; i < n2; i++) {
        memcpy(tmpBuf, inEqn->coeff2[i], 3 * sizeof(double));
        memcpy(inEqn->coeff2[i], inEqn->coeff2[n - i], 3 * sizeof(double));
        memcpy(inEqn->coeff2[n - i], tmpBuf, 3 * sizeof(double));
    }
    return;
}
/********************************************************************/
/*
 * copyBernParQuad( inEqn) - copy bernstein-parametric eqn, return pointer
 *
 */
/********************************************************************/
BernParQuad_Ptr
copyBernParQuad(inEqn)
    BernParQuad_Ptr inEqn;
{
    int             i;
    BernParQuad_Ptr eqn;
    if (inEqn == NULL) {
        return NULL;
    }
    eqn = (BernParQuad_Ptr) malloc(sizeof(BernParQuad));
    eqn->degree = inEqn->degree;
    if (eqn->degree > 0) {
        eqn->coeff1 = (double (*)[3])
            createMem(3 * (1 + eqn->degree) * sizeof(double));
        eqn->coeff2 = (double (*)[3])
            createMem(3 * (1 + eqn->degree) * sizeof(double));
        memcpy(eqn->coeff1,inEqn->coeff1,
              3 * (1 + eqn->degree) * sizeof(double));
        memcpy(eqn->coeff2,inEqn->coeff2,
              3 * (1 + eqn->degree) * sizeof(double));
    }
    return eqn;
}
/********************************************************************/
/*
 * reverseBernTensor( inEqn) - reverse bernstein-parametric quad eqn
 *
 */
/********************************************************************/
void
reverseBernTensor(inEqn)
    BernTensor_Ptr  inEqn;
{
    int             i;
    int                 n, n2;
    double              tmpBuf[3];
    if ((inEqn == NULL) || (inEqn->degree == 0)) {
        return;
    }
    n = (1 + inEqn->degree);
    n2 = n / 2;
    for (i = 0; i < n2; i++) {
        memcpy(tmpBuf, inEqn->coeff1[i], 3 * sizeof(double));
        memcpy(inEqn->coeff1[i], inEqn->coeff1[n - i], 3 * sizeof(double));
        memcpy(inEqn->coeff1[n - i], tmpBuf, 3 * sizeof(double));
    }
    for (i = 0; i < n2; i++) {
        memcpy(tmpBuf, inEqn->coeff2[i], 3 * sizeof(double));
        memcpy(inEqn->coeff2[i], inEqn->coeff2[n - i], 3 * sizeof(double));
        memcpy(inEqn->coeff2[n - i], tmpBuf, 3 * sizeof(double));
    }
    return;
}
/********************************************************************/
/*
 * copyBernTensor( inEqn) - copy bernstein-parametric eqn, return pointer
 *
 */
/********************************************************************/
BernTensor_Ptr
copyBernTensor(inEqn)
    BernTensor_Ptr  inEqn;
{
    int             i;
    BernTensor_Ptr  eqn;
    if (inEqn == NULL) {
        return NULL;
    }
    eqn = (BernTensor_Ptr) malloc(sizeof(BernTensor));
    eqn->degree = inEqn->degree;
    if (eqn->degree > 0) {
        eqn->coeff1 = (double (*)[3])
            createMem(3 * (1 + eqn->degree) * sizeof(double));
        eqn->coeff2 = (double (*)[3])
            createMem(3 * (1 + eqn->degree) * sizeof(double));
        memcpy(eqn->coeff1,inEqn->coeff1,
              3 * (1 + eqn->degree) * sizeof(double));
        memcpy(eqn->coeff2,inEqn->coeff2,
              3 * (1 + eqn->degree) * sizeof(double));
        memcpy(eqn->tangent,inEqn->tangent,
              3 * sizeof(double));
    }
    return eqn;
}

/********************************************************************/
/*
```

```
 * copyVertex(inVertex) - copy in and create a single vertex return a
     pointer
 * to the vertex
 */
/********************************************************************/
Vertex_Ptr
copyVertex(inVertex)
    Vertex_Ptr      inVertex;
{
    Vertex_Ptr      New_Vertex = createVertex();
    AdjList_Ptr     last_adj, src_adj;
    int             i, num_adj;
    double          a, b, c;

    /* copy in the point value */
    memcpy(New_Vertex->point, inVertex->point, sizeof(double) * 3);

    /* copy adjacencies */
    for (src_adj = inVertex->adjacencies, i = 0; src_adj != NULL;
         src_adj = src_adj->next, i++) {
        if (i == 0) {
            last_adj = New_Vertex->adjacencies = createAdjItem();
            copyAdjItem(src_adj, last_adj);
        } else {
            last_adj->next = createAdjItem();
            copyAdjItem(src_adj, last_adj->next);
            last_adj = last_adj->next;
        }
    }
    return (New_Vertex);
}

/********************************************************************/
/*
 * copyDEdge(inDEdge) - copy in and create a new directed edge
 *
 */
/********************************************************************/
DEdge_Ptr
copyDEdge(inDEdge)
    DEdge_Ptr       inDEdge;
{
    DEdge_Ptr       New_DEdge = createDEdge();

    copyIndex(&inDEdge->cycle, &New_DEdge->cycle);
    New_DEdge->rightOrientation = inDEdge->rightOrientation;
    copyIndex(&inDEdge->edge, &New_DEdge->edge);
    copyIndex(&inDEdge->nextDE, &New_DEdge->nextDE);
    return (New_DEdge);
}

/********************************************************************/
/*
```

```
 * copyEdge(inEdge) - copy in and create an edge return a pointer to the
     edge
 *
 */
/*********************************************************************/
Edge_Ptr
copyEdge(inEdge)
    Edge_Ptr        inEdge;
{
    Edge_Ptr        New_Edge = createEdge();
    DEList_Ptr      last_de, src_de;
    int             i;

    /* copy edge name */
    strcpy(New_Edge->name, inEdge->name);

    /* copy vertex1 & vertex2 indices */
    copyIndex(&inEdge->vertex1, &New_Edge->vertex1);
    copyIndex(&inEdge->vertex2, &New_Edge->vertex2);

    /* copy edge type */
    New_Edge->type = inEdge->type;

    /* copy tangents */
    memcpy(New_Edge->tan12, inEdge->tan12, sizeof(double) * 3);
    memcpy(New_Edge->tan21, inEdge->tan21, sizeof(double) * 3);

    /* copy directed edges */
    for (src_de = inEdge->dEdges, i = 0; src_de != NULL;
         src_de = src_de->next, i++) {
        if (i == 0) {
            last_de = New_Edge->dEdges = createDEdgeItem();
            copyIndex(&src_de->dEdge, &last_de->dEdge);
        } else {
            last_de->next = createDEdgeItem();
            copyIndex(&src_de->dEdge, &last_de->next->dEdge);
            last_de = last_de->next;
        }
    }

    /* copy aux eqn */
    New_Edge->aux_Eqn = CopyPoly(inEdge->aux_Eqn);

    /* see if there is a bernstein eqn */
    New_Edge->eqn = copyBernPar(inEdge->eqn);
    return (New_Edge);
}

/*********************************************************************/
/*
 * copyCycle(inCycle) - copy in, create and return a cycle
 *
 */
```

```
/*********************************************************************/
Cycle_Ptr
copyCycle(inCycle)
    Cycle_Ptr       inCycle;
{
    Cycle_Ptr       New_Cycle = createCycle();

    copyIndex(&inCycle->face, &New_Cycle->face);
    copyIndex(&inCycle->dEdge, &New_Cycle->dEdge);

    return (New_Cycle);
}

/*********************************************************************/
/*
 * copyFace(inFace) - copy in and create a face return a pointer to the new
 * face
 *
 */
/*********************************************************************/
Face_Ptr
copyFace(inFace)
    Face_Ptr        inFace;
{
    Face_Ptr        New_Face = createFace();
    EQNList_Ptr     last_eqn, next_eqn;
    CycleList_Ptr   last_cycle, src_cycle;
    int             i;

    /* copy name */
    strcpy(New_Face->name, inFace->name);

    /* copy type */
    New_Face->type = inFace->type;

    /* copy equation */
    New_Face->equation = CopyPoly(inFace->equation);
    New_Face->bernQuad = copyBernParQuad(inFace->bernQuad);
    New_Face->bernTens = copyBernTensor(inFace->bernTens);

    /* copy the (three) normal equations */
    New_Face->normal = copyEqnItem(inFace->normal);
    New_Face->normal->next = copyEqnItem(inFace->normal->next);
    New_Face->normal->next->next = copyEqnItem(inFace->normal->next->next);

    /* copy in the cycles */
    for (src_cycle = inFace->cycles, i = 0; src_cycle != NULL;
         src_cycle = src_cycle->next, i++) {
        if (i == 0) {
            last_cycle = New_Face->cycles = createCycleItem();
            copyIndex(&src_cycle->cycle, &last_cycle->cycle);
        } else {
            last_cycle->next = createCycleItem();
            copyIndex(&src_cycle->cycle, &last_cycle->next->cycle);
            last_cycle = last_cycle->next;
        }
    }

    return (New_Face);
}

/*********************************************************************/
/*
 * copySolid(inSolid) - copy a solid from another. return a pointer to the
 * new solid
 *
 */
/*********************************************************************/
Solid_Ptr
copySolid(inSolid)
    Solid_Ptr       inSolid;
{
    /* WARNING-- if marked field is -1, piece won't be copied */
    Solid_Ptr       New_Solid = createSolid();
    int             i;
    Stack_Union     object;

    strcpy(New_Solid->name, inSolid->name);

    /* copy all the solid subcomponents */
    printf("copying vertices\n");
    for (i = 0; i < inSolid->vertices->index; i++) {
        object.vertex = copyVertex(Solid_Vertex(inSolid, i));
        AddObjToSolid(&object, VERTEX, New_Solid);
    }

    printf("copying edges\n");
    for (i = 0; i < inSolid->edges->index; i++) {
        object.edge = copyEdge(Solid_Edge(inSolid, i));
        AddObjToSolid(&object, EDGE, New_Solid);
    }

    printf("copying faces\n");
    for (i = 0; i < inSolid->faces->index; i++) {
        object.face = copyFace(Solid_Face(inSolid, i));
        AddObjToSolid(&object, FACE, New_Solid);
    }

    printf("copying dedges\n");
    for (i = 0; i < inSolid->dEdges->index; i++) {
        object.dEdge = copyDEdge(Solid_DEdge(inSolid, i));
        AddObjToSolid(&object, DEDGE, New_Solid);
    }

    printf("copying cycles\n");
    for (i = 0; i < inSolid->cycles->index; i++) {
        object.cycle = copyCycle(Solid_Cycle(inSolid, i));
        AddObjToSolid(&object, CYCLE, New_Solid);
    }

    return (New_Solid);
}

/**********************************************************************/
/*
 * copyMarkedSolid(inSolid) - copy a marked solid from another. return a
 * pointer to the new solid, marked fields not copied
 *
 */
/**********************************************************************/
Solid_Ptr
copyMarkedSolid(inSolid)
    Solid_Ptr       inSolid;
{
    Solid_Ptr       New_Solid = createSolid();
    int             i;
    Stack_Union     object;
    int             nfv, nfe, nff, nfc, nfd;

    strcpy(New_Solid->name, inSolid->name);

    nfv = inSolid->vertices->index;
    nfe = inSolid->edges->index;
    nff = inSolid->faces->index;
    nfc = inSolid->cycles->index;
    nfd = inSolid->dEdges->index;

    printf("copying unmarked vertices\n");
    for (i = 0; i < nfv; i++) {
        AdjList_Ptr     adjs;
        Vertex_Ptr      V, fV;
        Face_Ptr        fF;
        DEdge_Ptr       fD;
        int             iV;

        fV = Solid_Vertex(inSolid, i);
        if (fV->marked == -1) {
            continue;
        }
        V = object.vertex = copyVertex(fV);
        AddObjToSolid(&object, VERTEX, New_Solid);

        for (adjs = V->adjacencies; adjs != NULL; adjs = adjs->next) {
            fF = Solid_Face(inSolid, adjs->face.index - 1);
            if (fF->marked == -1) {
                fprintf(stderr, "copyMarkedSolid()->Warning: bad face %d on
                    adjs!\n",
                    adjs->face.index - 1);
            } else {
                adjs->face.index -= fF->marked;
            }

            fD = Solid_DEdge(inSolid, adjs->dEIn.index - 1);
            if (fD->marked == -1) {
                fprintf(stderr, "copyMarkedSolid()->Warning: bad deIn %d in
                    adjs!\n",
                    adjs->dEIn.index - 1);
            } else {
                adjs->dEIn.index -= fD->marked;
            }

            fD = Solid_DEdge(inSolid, adjs->dEOut.index - 1);
            if (fD->marked == -1) {
                fprintf(stderr, "copyMarkedSolid()->Warning: bad deOut %d
                    in adjs!\n",
                    adjs->dEOut.index - 1);
            } else {
                adjs->dEOut.index -= fD->marked;
            }
        }
    }

    printf("copying unmarked edges\n");
    for (i = 0; i < nfe; i++) {
        Edge_Ptr        E, fE;
        Vertex_Ptr      fV;
        DEList_Ptr      des;
        int             iE;

        fE = Solid_Edge(inSolid, i);
        if (fE->marked == -1) {
            continue;
        }
        E = object.edge = copyEdge(fE);
        AddObjToSolid(&object, EDGE, New_Solid);

        fV = Solid_Vertex(inSolid, E->vertex1.index - 1);
        if (fV->marked == -1) {
            fprintf(stderr, "copyMarkedSolid()->Warning: bad vert %d on
                edge!\n",
                E->vertex1.index - 1);
        } else {
            E->vertex1.index -= fV->marked;
        }

        fV = Solid_Vertex(inSolid, E->vertex2.index - 1);
        if (fV->marked == -1) {
            fprintf(stderr, "copyMarkedSolid()->Warning: bad vert %d on
                edge!\n",
                E->vertex2.index - 1);
        } else {
            E->vertex2.index -= fV->marked;
        }

        for (des = E->dEdges; des != NULL; des = des->next) {
            DEdge_Ptr       fD;
            fD = Solid_DEdge(inSolid, des->dEdge.index - 1);
            if (fD->marked == -1) {
                fprintf(stderr, "copyMarkedSolid()->Warning: bad dedge %d
                    on edge!\n",
                    des->dEdge.index - 1);
            } else {
                des->dEdge.index -= fD->marked;
            }
        }
    }

    printf("copying unmarked faces\n");
    for (i = 0; i < nff; i++) {
        Face_Ptr        F, fF;
        CycleList_Ptr   cycs;

        fF = Solid_Face(inSolid, i);
        if (fF->marked == -1) {
            continue;
        }
        F = object.face = copyFace(fF);
        AddObjToSolid(&object, FACE, New_Solid);

        for (cycs = F->cycles; cycs != NULL; cycs = cycs->next) {
            Cycle_Ptr       fC;

            fC = Solid_Cycle(inSolid, cycs->cycle.index - 1);
            if (fC->marked == -1) {
                fprintf(stderr, "copyMarkedSolid()->Warning: bad cyc %d on
                    face!\n",
                    cycs->cycle.index - 1);
            } else {
                cycs->cycle.index -= fC->marked;
            }
        }
    }

    printf("copying unmarked dedges\n");
    for (i = 0; i < nfd; i++) {
        DEdge_Ptr       D, fD;
        Cycle_Ptr       fC;
        Edge_Ptr        fE;
        DEdge_Ptr       fDn;

        fD = Solid_DEdge(inSolid, i);
        if (fD->marked == -1) {
            continue;
        }
        D = object.dEdge = copyDEdge(fD);
        AddObjToSolid(&object, DEDGE, New_Solid);

        fC = Solid_Cycle(inSolid, D->cycle.index - 1);
        if (fC->marked == -1) {
            fprintf(stderr, "copyMarkedSolid()->Warning: bad cycle %d on
                dedge!\n",
                D->cycle.index - 1);
        } else {
            D->cycle.index -= fC->marked;
        }

        fE = Solid_Edge(inSolid, D->edge.index - 1);
        if (fE->marked == -1) {
            fprintf(stderr, "copyMarkedSolid()->Warning: bad edge %d of
                dedge!\n",
                D->edge.index - 1);
        } else {
            D->edge.index -= fE->marked;
        }

        fD = Solid_DEdge(inSolid, D->nextDE.index - 1);
        if (fD->marked == -1) {
            fprintf(stderr, "copyMarkedSolid()->Warning: bad nextDE %d in
                dedge!\n",
                D->nextDE.index - 1);
        } else {
            D->nextDE.index -= fD->marked;
        }
    }

    printf("copying unmarked cycles\n");
    for (i = 0; i < nfc; i++) {
        Cycle_Ptr       C, fC;
        Face_Ptr        fF;
        DEdge_Ptr       fD;

        fC = Solid_Cycle(inSolid, i);
        if (fC->marked == -1) {
            continue;
        }
        C = object.cycle = copyCycle(fC);
        AddObjToSolid(&object, CYCLE, New_Solid);

        fF = Solid_Face(inSolid, C->face.index - 1);
        if (fF->marked == -1) {
            fprintf(stderr, "copyMarkedSolid()->Warning: bad face %d on
                cycle!\n",
                C->face.index - 1);
        } else {
            C->face.index -= fF->marked;
        }

        fD = Solid_DEdge(inSolid, C->dEdge.index - 1);
        if (fD->marked == -1) {
            fprintf(stderr, "copyMarkedSolid()->Warning: bad de %d in cycle
                !\n",
                C->dEdge.index - 1);
        } else {
            C->dEdge.index -= fD->marked;
        }
    }
    return New_Solid;
}
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
#include <stdio.h>
#include <math.h>
#include <ctype.h>

#include <shastra/network/server.h>
#include <shastra/network/mplex.h>

#include <shastra/draw/image.h>
#include <shastra/draw/drawdata.h>
#include <shastra/solid/imageIO.h>


static int      fUseNormals = 0;
static int      fCWPolys = 1;

void            normalizeNormal(Prot1(float *));

mPolygonData   *
readPolyImageFD(fd)
     int             fd;
{
  int             i, j;

  mPolygonData   *mPoly;
  polygonData    *poly;
  char           *sbIn;

  sbIn = cmReceiveString(fd);             /*WPZ*/
  if( sbIn[0] == '\0') return NULL;       /*WPZ*/
  mPoly = (mPolygonData *) malloc(sizeof(mPolygonData));

  sscanf(sbIn, "%d", &mPoly->nPolygons);
  free(sbIn);
  mPoly->polygons = (polygonData *) malloc(sizeof(polygonData) *
                      mPoly->nPolygons);
  memset(mPoly->polygons,0, sizeof(polygonData) *mPoly->nPolygons);
  for (i = 0; i < mPoly->nPolygons; i++) {
    poly = &mPoly->polygons[i];
    sbIn = cmReceiveString(fd);
    sscanf(sbIn, "%d", &poly->nPoints);
    free(sbIn);
    poly->array = (double (*)[3]) malloc(sizeof(double) *
                     3 * poly->nPoints);
    poly->normals = (float (*)[3]) malloc(sizeof(float) *
                     3 * poly->nPoints);

    for (j = 0; j < poly->nPoints; j++) {
      sbIn = cmReceiveString(fd);
      if (fUseNormals) {
    sscanf(sbIn, "%lf%lf%lf%f%f%f",
           &poly->array[j][0],
           &poly->array[j][1],
           &poly->array[j][2],
           &poly->normals[j][0],
           &poly->normals[j][1],
           &poly->normals[j][2]);
      } else {
    sscanf(sbIn, "%lf%lf%lf",
           &poly->array[j][0],
           &poly->array[j][1],
           &poly->array[j][2]);
      }
      free(sbIn);
    }
  }
  if (!fUseNormals) {
    computeImageNormals(mPoly);
  }
  return mPoly;
}
mPolygonData   *
readPolyImage(inStream)
     FILE           *inStream;
{
  int             i, j;

  mPolygonData   *mPoly;
  polygonData    *poly;

  mPoly = (mPolygonData *) malloc(sizeof(mPolygonData));
  fscanf(inStream, "%d", &mPoly->nPolygons);
  mPoly->polygons = (polygonData *) malloc(sizeof(polygonData) *
                     mPoly->nPolygons);
  memset(mPoly->polygons,0,sizeof(polygonData) *mPoly->nPolygons);
  for (i = 0; i < mPoly->nPolygons; i++) {
    poly = &mPoly->polygons[i];
    fscanf(inStream, "%d", &poly->nPoints);
    poly->array = (double (*)[3]) malloc(sizeof(double) *
                   3 * poly->nPoints);
    poly->normals = (float (*)[3]) malloc(sizeof(float) *
                    3 * poly->nPoints);

    for (j = 0; j < poly->nPoints; j++) {
      if (fUseNormals) {
    fscanf(inStream, "%lf%lf%lf%f%f%f",
           &poly->array[j][0],
           &poly->array[j][1],
           &poly->array[j][2],
           &poly->normals[j][0],
           &poly->normals[j][1],
           &poly->normals[j][2]);
      } else {
    fscanf(inStream, "%lf%lf%lf",
           &poly->array[j][0],
           &poly->array[j][1],
           &poly->array[j][2]);
      }
    }
  }
  if (!fUseNormals) {
    computeImageNormals(mPoly);
  }
  return mPoly;
}

void
writePolyImageFD(fd, mPoly)
     int              fd;
     mPolygonData   *mPoly;
{
  FILE           *outStream;
  int             i, j;
  polygonData    *poly;
  char            sbOut[256];

  sprintf(sbOut, "%d\n", mPoly->nPolygons);
  cmSendString(fd, sbOut);
  for (i = 0; i < mPoly->nPolygons; i++) {
    poly = &mPoly->polygons[i];
    sprintf(sbOut, "%d\n", poly->nPoints);
    cmSendString(fd, sbOut);

    for (j = 0; j < poly->nPoints; j++) {
      if (fUseNormals) {
```

```
    sprintf(sbOut, "%lf %lf %lf %f %f %f\n",
        poly->array[j][0],
        poly->array[j][1],
        poly->array[j][2],
        poly->normals[j][0],
        poly->normals[j][1],
        poly->normals[j][2]);
      } else {
    sprintf(sbOut, "%lf %lf %lf\n",
        poly->array[j][0],
        poly->array[j][1],
        poly->array[j][2]);
      }
      cmSendString(fd, sbOut);
    }
  }
}

void
writePolyImage(outStream, mPoly)
     FILE          *outStream;
     mPolygonData   *mPoly;
{
  int             i, j;
  polygonData    *poly;

  fprintf(outStream, "%d\n", mPoly->nPolygons);
  for (i = 0; i < mPoly->nPolygons; i++) {
    poly = &mPoly->polygons[i];
    fprintf(outStream, "%d\n", poly->nPoints);

    for (j = 0; j < poly->nPoints; j++) {
      if (fUseNormals) {
    fprintf(outStream, "%lf %lf %lf %f %f %f\n",
        poly->array[j][0],
        poly->array[j][1],
        poly->array[j][2],
        poly->normals[j][0],
        poly->normals[j][1],
        poly->normals[j][2]);
      } else {
    fprintf(outStream, "%lf %lf %lf\n",
        poly->array[j][0],
        poly->array[j][1],
        poly->array[j][2]);
      }
    }
  }
}

void
freePolyImage(mPoly)
     mPolygonData   *mPoly;
```

```
{
  int             i, j;
  polygonData    *poly;

  for (i = 0; i < mPoly->nPolygons; i++) {
    poly = &mPoly->polygons[i];
    free(poly->array);
    free(poly->normals);
    if(poly->scratch){
      free(poly->scratch);
    }
  }
  free(mPoly->polygons);
  free(mPoly);
}

computeImageNormals(mPoly)
     mPolygonData   *mPoly;
{
  int             i, j;
  polygonData    *poly;
  int             jj1, jj2;
  for (i = 0; i < mPoly->nPolygons; i++) {
    poly = &mPoly->polygons[i];
    if (poly->nPoints < 3) {
      fprintf(stderr, "computeImageNormals()-- poly has < 3pts\n");
    }
    for (j = 0; j < poly->nPoints; j++) {
      jj1 = j + 1;
      if (jj1 >= poly->nPoints) {
    jj1 -= poly->nPoints;
      }
      jj2 = j + 2;
      if (jj2 >= poly->nPoints) {
    jj2 -= poly->nPoints;
      }
      if (fCWPolys) {   /* clockwise */
    if (PlaneNormalFrom3Pts(poly->array[j], poly->array[jj1],
                poly->array[jj2], poly->normal) == 1) {
      break;
    }
      } else {/* counterclockwise */
    if (PlaneNormalFrom3Pts(poly->array[jj2], poly->array[jj1],
                poly->array[j], poly->normal) == 1) {
      break;
    }
      }
    }
    if (j == poly->nPoints) {
      fprintf(stderr, "computeImageNormals()-- poly pts are collinear\n");
    }
    /* flat shaded for now */
    for (j = 0; j < poly->nPoints; j++) {
      memcpy(poly->normals[j], poly->normal, sizeof(float) * 3);
    }
  }
}

mPolygonData   *
readPolyImageNoCount(inStream)
     FILE           *inStream;
{
  int               i, j;

  mPolygonData   *mPoly;
  polygonData    *poly;
  int               nPolygons = 1024;

  mPoly = (mPolygonData *) malloc(sizeof(mPolygonData));
  mPoly->polygons = (polygonData *) malloc(sizeof(polygonData) *
                       nPolygons);
  memset(mPoly->polygons,0,sizeof(polygonData) *mPoly->nPolygons);
  mPoly->nPolygons = 0;
  i = 0;
  while (1) {
    if (i == nPolygons) {
      nPolygons *= 2;
      mPoly->polygons = (polygonData *) realloc(mPoly->polygons,
                        sizeof(polygonData) * nPolygons);
      memset(&mPoly->polygons[nPolygons/2], 0,sizeof(polygonData) *
          nPolygons/2);
    }
    poly = &mPoly->polygons[i];
    if (fscanf(inStream, "%d", &poly->nPoints) == EOF) {
      break;
    }
    mPoly->nPolygons++;
    i++;
    poly->array = (double (*)[3]) malloc(sizeof(double) *
                     3 * poly->nPoints);
    poly->normals = (float (*)[3]) malloc(sizeof(float) *
                      3 * poly->nPoints);

    for (j = 0; j < poly->nPoints; j++) {
      if (fUseNormals) {
    fscanf(inStream, "%lf%lf%lf%lf%lf%lf",
           &poly->array[j][0],
           &poly->array[j][1],
           &poly->array[j][2],
           &poly->normals[j][0],
           &poly->normals[j][1],
           &poly->normals[j][2]);
      } else {
    fscanf(inStream, "%lf%lf%lf",
           &poly->array[j][0],
           &poly->array[j][1],
              &poly->array[j][2]);
        }
      }
    }
    if (!fUseNormals) {
      computeImageNormals(mPoly);
    }
    mPoly->polygons = (polygonData *) realloc(mPoly->polygons, mPoly->
        nPolygons *
                          sizeof(polygonData));
    return mPoly;
  }

  mPolygonData   *
  copyPolyImage(inmPoly)
       mPolygonData   *inmPoly;
  {
    int             i, j;

    mPolygonData   *mPoly;
    polygonData    *poly;
    polygonData    *inpoly;

    mPoly = (mPolygonData *) malloc(sizeof(mPolygonData));
    mPoly->nPolygons = inmPoly->nPolygons;
    mPoly->polygons = (polygonData *) malloc(sizeof(polygonData) *
                         mPoly->nPolygons);
    for (i = 0; i < mPoly->nPolygons; i++) {
      poly = &mPoly->polygons[i];
      inpoly = &inmPoly->polygons[i];
      poly->nPoints = inpoly->nPoints;
      poly->array = (double (*)[3]) malloc(sizeof(double) *
                        3 * poly->nPoints);
      poly->normals = (float (*)[3]) malloc(sizeof(float) *
                         3 * poly->nPoints);

      memcpy(poly->array, inpoly->array, sizeof(double) *
         3 * poly->nPoints);
      memcpy(poly->normals, inpoly->normals, sizeof(double) *
         3 * poly->nPoints);
    }
    return mPoly;
  }

  void
  setPolyNormMode(mode)
       int             mode;
  {
    fUseNormals = mode;
  }

  void
  setPolyOrientMode(mode)
```

```
     int                mode;
{
  fCWPolys = mode;
}

int
getPolyNormMode()
{
  return fUseNormals;
}

int
getPolyOrientMode()
{
  return fCWPolys;
}

int
PlaneNormalFrom3Pts(v1, v2, v3, norm)
     double             v1[3], v2[3], v3[3];
     float norm[3];
{
  double          u[3], v[3], A, B, C, D;
  int             i;

  for (i = 0; i < 3; i++) {
    u[i] = v1[i] - v2[i];
    v[i] = v3[i] - v2[i];
  }
  A = u[1] * v[2] - v[1] * u[2];
  B = u[2] * v[0] - u[0] * v[2];
  C = u[0] * v[1] - u[1] * v[0];
  D = -(A * v1[0] + B * v1[1] + C * v1[2]);

  norm[0] = A;
  norm[1] = B;
  norm[2] = C;
  /* check if the three points were collinear */
  if ((fabs(A) == 0.0) && (fabs(B) == 0.0) && (fabs(C) == 0.0)) {

    fprintf(stderr, " PlaneNormalFrom3Pts()->collinear points!\n");
    fprintf(stderr, "[0] %lf %lf %lf [1] %lf %lf %lf [2] %lf %lf %lf\n",
        v1[0],v1[1],v1[2],v2[0],v2[1],v2[2], v3[0],v3[1],v3[2]);
    fprintf(stderr, "  set plane normal to (0,0,1)\n");
    norm[0] = 0;
    norm[1] = 0;
    norm[2] = 1;
    return (0);
  }
  normalizeNormal(norm);

  return (1);
}
```

```
void
normalizeNormal(pNormal)
     float         *pNormal;
{
  double          tmpSum;
  int             i;

  tmpSum = 0.0;
  for (i = 0; i < 3; i++) {
    tmpSum += pNormal[i] * pNormal[i];
  }
  tmpSum = sqrt(tmpSum);
  for (i = 0; i < 3; i++) {
    pNormal[i] = pNormal[i] / tmpSum;
  }
}

void
normalizeDblVector(pNormal)
     double         *pNormal;
{
  double          tmpSum;
  int             i;

  tmpSum = 0.0;
  for (i = 0; i < 3; i++) {
    tmpSum += pNormal[i] * pNormal[i];
  }
  tmpSum = sqrt(tmpSum);
  for (i = 0; i < 3; i++) {
    pNormal[i] = pNormal[i] / tmpSum;
  }
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
#include <stdio.h>
#include <ctype.h>
#include <shastra/solid/indexPolyH.h>
#include <shastra/network/mplex.h>
#include <shastra/network/rpc.h>
#include <shastra/network/server.h>

#define STANDALONEnn

static char         sbOut[5120];

int
IndexPolyOut(fd, pIPoly)
    int             fd;
    IndexPoly      *pIPoly;
{
    XDR             xdrs;
    int             retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdout /* fdopen(fd,"w") */ ;
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
        if (!xdr_IndexPoly(&xdrs, pIPoly)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
```

```
    /*
     * xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
     */
    if (!xdr_IndexPoly(mplexXDRSEnc(fd), pIPoly)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
IndexPolyIn(fd, pIPoly)
    int             fd;
    IndexPoly      *pIPoly;
{
    XDR             xdrs;
    int             retVal = 0;

    IndexPolyXDRFree(pIPoly);
#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdin /* fdopen(fd,"r") */ ;
        xdrstdio_create(&xdrs, fp, XDR_DECODE);
        if (!xdr_IndexPoly(&xdrs, pIPoly)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);
     */
    if (!xdr_IndexPoly(mplexXDRSDec(fd), pIPoly)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}



void
inputIndexPoly(fp, pIPoly)
    FILE           *fp;
    IndexPoly      *pIPoly;
{
    int             i,j;

    fscanf(fp, "%u", &pIPoly->vertices.vertices_len);
    pIPoly->vertices.vertices_val =
        (IndexPolyVert *) malloc(sizeof(IndexPolyVert) *
                         pIPoly->vertices.vertices_len);
    for (i = 0; i < pIPoly->vertices.vertices_len; i++) {
        fscanf(fp, "%lf%lf%lf",
               &pIPoly->vertices.vertices_val[i][0],
               &pIPoly->vertices.vertices_val[i][1],
               &pIPoly->vertices.vertices_val[i][2]);
    }

    fscanf(fp, "%u", &pIPoly->edgeVerts.edgeVerts_len);
    pIPoly->edgeVerts.edgeVerts_val =
        (IndexPolyEdge *) malloc(sizeof(IndexPolyEdge) *
                     pIPoly->edgeVerts.edgeVerts_len);
    for (i = 0; i < pIPoly->edgeVerts.edgeVerts_len; i++) {
        fscanf(fp, "%d%d",
               &pIPoly->edgeVerts.edgeVerts_val[i][0],
               &pIPoly->edgeVerts.edgeVerts_val[i][1]);
    }

    fscanf(fp, "%u", &pIPoly->faces.faces_len);
    pIPoly->faces.faces_val =
        (faceEdges *) malloc(sizeof(faceEdges) *
                     pIPoly->faces.faces_len);
    for (i = 0; i < pIPoly->faces.faces_len; i++) {
        fscanf(fp, "%u", &pIPoly->faces.faces_val[i].faceEdges_len);
        pIPoly->faces.faces_val[i].faceEdges_val =
            (int *) malloc(sizeof(int) *
                  pIPoly->faces.faces_val[i].faceEdges_len);
        for (j = 0; j < pIPoly->faces.faces_val[i].faceEdges_len; j++) {
            fscanf(fp, "%d",
                   &pIPoly->faces.faces_val[i].faceEdges_val[j]);
        }
    }
}

void
outputIndexPoly(fp, pIPoly)
    FILE               *fp;
    IndexPoly          *pIPoly;
{
    int                 i,j;

    fprintf(fp, "%u\n", pIPoly->vertices.vertices_len);
    for (i = 0; i < pIPoly->vertices.vertices_len; i++) {
        fprintf(fp, "%lf %lf %lf\n",
            pIPoly->vertices.vertices_val[i][0],
            pIPoly->vertices.vertices_val[i][1],
            pIPoly->vertices.vertices_val[i][2]);
    }

    fprintf(fp, "%u\n", pIPoly->edgeVerts.edgeVerts_len);
    for (i = 0; i < pIPoly->edgeVerts.edgeVerts_len; i++) {
        fprintf(fp, "%d %d\n",
            pIPoly->edgeVerts.edgeVerts_val[i][0],
            pIPoly->edgeVerts.edgeVerts_val[i][1]);
    }
    fprintf(fp, "%u\n", pIPoly->faces.faces_len);
    for (i = 0; i < pIPoly->faces.faces_len; i++) {
        fprintf(fp, "%u\n", pIPoly->faces.faces_val[i].faceEdges_len);
        for (j = 0; j < pIPoly->faces.faces_val[i].faceEdges_len; j++) {
            fprintf(fp, "%d ",
                pIPoly->faces.faces_val[i].faceEdges_val[j]);
        }
        fprintf(fp, "\n");
    }
}



void
freeIndexPoly(pIPoly)
    IndexPoly      *pIPoly;
{
    int             i;

    free(pIPoly->vertices.vertices_val);
    free(pIPoly->edgeVerts.edgeVerts_val);

    for (i = 0; i < pIPoly->faces.faces_len; i++) {
        free(pIPoly->faces.faces_val[i].faceEdges_val);
    }
    free(pIPoly->faces.faces_val);
    memset(pIPoly, 0, sizeof(IndexPoly));
}

IndexPoly      *
copyIndexPoly(pIPoly, destpIPoly)
    IndexPoly      *pIPoly;
    IndexPoly      *destpIPoly;
{
    IndexPoly      *newpIPoly;
    int             i;

    if (pIPoly == NULL) {
        return NULL;
    }
    if (destpIPoly == NULL) {
        newpIPoly = (IndexPoly *) malloc(sizeof(IndexPoly));
    } else {
        newpIPoly = destpIPoly;
    }

    destpIPoly->vertices.vertices_len = pIPoly->vertices.vertices_len;
    destpIPoly->vertices.vertices_val =
        (IndexPolyVert *) malloc(sizeof(IndexPolyVert) *
                    pIPoly->vertices.vertices_len);
    memcpy(destpIPoly->vertices.vertices_val,pIPoly->vertices.vertices_val,
          sizeof(IndexPolyVert) *
```

```
            pIPoly->vertices.vertices_len);

    destpIPoly->edgeVerts.edgeVerts_len = pIPoly->edgeVerts.edgeVerts_len;
    destpIPoly->edgeVerts.edgeVerts_val =
        (IndexPolyEdge *) malloc(sizeof(IndexPolyEdge) *
                     pIPoly->edgeVerts.edgeVerts_len);
    memcpy( destpIPoly->edgeVerts.edgeVerts_val,
            pIPoly->edgeVerts.edgeVerts_val,
          sizeof(IndexPolyEdge) * pIPoly->edgeVerts.edgeVerts_len);

    destpIPoly->faces.faces_len = pIPoly->faces.faces_len;
    destpIPoly->faces.faces_val =
        (faceEdges *) malloc(sizeof(faceEdges) *
                     pIPoly->faces.faces_len);
    for (i = 0; i < pIPoly->faces.faces_len; i++) {
        destpIPoly->faces.faces_val[i].faceEdges_len =
            pIPoly->faces.faces_val[i].faceEdges_len;
        destpIPoly->faces.faces_val[i].faceEdges_val =
            (int *) malloc(sizeof(int) *
                  pIPoly->faces.faces_val[i].faceEdges_len);
        memcpy( destpIPoly->faces.faces_val[i].faceEdges_val,
                pIPoly->faces.faces_val[i].faceEdges_val,
            sizeof(int) * pIPoly->faces.faces_val[i].faceEdges_len);
    }
    return destpIPoly;
}

void
IndexPolyXDRFree(pIPoly)
    IndexPoly      *pIPoly;
{
    xdr_free(xdr_IndexPoly, (char *) pIPoly);
    memset(pIPoly, 0, sizeof(IndexPoly));
}





IndexPoly      *
inputIPolyString(fd)
    int             fd;
{
    IndexPoly      *pIPoly;
    int             i,j;
    char *sbIn;

    pIPoly = (IndexPoly*)malloc(sizeof(IndexPoly));
    memset(pIPoly, 0,sizeof(IndexPoly));
    sbIn = cmReceiveString(fd);
    sscanf(sbIn, "%u", &pIPoly->vertices.vertices_len);
    free(sbIn);
    pIPoly->vertices.vertices_val =
            (IndexPolyVert *) malloc(sizeof(IndexPolyVert) *
                             pIPoly->vertices.vertices_len);
        for (i = 0; i < pIPoly->vertices.vertices_len; i++) {
            sbIn = cmReceiveString(fd);
            sscanf(sbIn, "%lf%lf%lf",
                   &pIPoly->vertices.vertices_val[i][0],
                   &pIPoly->vertices.vertices_val[i][1],
                   &pIPoly->vertices.vertices_val[i][2]);
            free(sbIn);
        }

        sbIn = cmReceiveString(fd);
        sscanf(sbIn, "%u", &pIPoly->edgeVerts.edgeVerts_len);
        free(sbIn);
        pIPoly->edgeVerts.edgeVerts_val =
            (IndexPolyEdge *) malloc(sizeof(IndexPolyEdge) *
                             pIPoly->edgeVerts.edgeVerts_len);
        for (i = 0; i < pIPoly->edgeVerts.edgeVerts_len; i++) {
            sbIn = cmReceiveString(fd);
            sscanf(sbIn, "%d%d",
                   &pIPoly->edgeVerts.edgeVerts_val[i][0],
                   &pIPoly->edgeVerts.edgeVerts_val[i][1]);
            free(sbIn);
        }

        sbIn = cmReceiveString(fd);
        sscanf(sbIn, "%u", &pIPoly->faces.faces_len);
        free(sbIn);
        pIPoly->faces.faces_val =
            (faceEdges *) malloc(sizeof(faceEdges) *
                             pIPoly->faces.faces_len);
        for (i = 0; i < pIPoly->faces.faces_len; i++) {
            char *iptr;
            sbIn = cmReceiveString(fd);
            sscanf(sbIn, "%u", &pIPoly->faces.faces_val[i].faceEdges_len);
            free(sbIn);
            pIPoly->faces.faces_val[i].faceEdges_val =
                (int *) malloc(sizeof(int) *
                      pIPoly->faces.faces_val[i].faceEdges_len);
            iptr = sbIn = cmReceiveString(fd);
            for (j = 0; j < pIPoly->faces.faces_val[i].faceEdges_len; j++) {
                while((!isdigit(*iptr)) && (*iptr!='-')){
                    iptr++/*skip nonnumerics*/;
                }
                sscanf(iptr, "%d",
                       &pIPoly->faces.faces_val[i].faceEdges_val[j]);
                if(*iptr == '-'){
                    iptr++;
                }
                while(isdigit(*iptr))iptr++/*skip numerics*/;
            }
            free(sbIn);
        }
```

```
    return pIPoly;
}

void
outputIPolyString(fd, pIPoly)
    int     fd;
    IndexPoly      *pIPoly;
{
    int             i,j;

    sprintf(sbOut, "%u\n", pIPoly->vertices.vertices_len);
    cmSendString(fd,sbOut);
    for (i = 0; i < pIPoly->vertices.vertices_len; i++) {
        sprintf(sbOut, "%lf %lf %lf\n",
            pIPoly->vertices.vertices_val[i][0],
            pIPoly->vertices.vertices_val[i][1],
            pIPoly->vertices.vertices_val[i][2]);
        cmSendString(fd,sbOut);
    }

    sprintf(sbOut, "%u\n", pIPoly->edgeVerts.edgeVerts_len);
    cmSendString(fd,sbOut);
    for (i = 0; i < pIPoly->edgeVerts.edgeVerts_len; i++) {
        sprintf(sbOut, "%d %d\n",
            pIPoly->edgeVerts.edgeVerts_val[i][0],
            pIPoly->edgeVerts.edgeVerts_val[i][1]);
        cmSendString(fd,sbOut);
    }

    sprintf(sbOut, "%u\n", pIPoly->faces.faces_len);
    cmSendString(fd,sbOut);
    for (i = 0; i < pIPoly->faces.faces_len; i++) {
        char *optr;
        sprintf(sbOut, "%u\n", pIPoly->faces.faces_val[i].faceEdges_len);
        cmSendString(fd,sbOut);
        optr = sbOut;
        for (j = 0; j < pIPoly->faces.faces_val[i].faceEdges_len; j++) {
            sprintf(optr, "%d ",
                pIPoly->faces.faces_val[i].faceEdges_val[j]);
            optr += strlen(optr);
        }
        sprintf(optr, "\n");
        cmSendString(fd,sbOut);
    }
}




#ifdef STANDALONE
main(argc, argv)
#else               /* STANDALONE */
```

```
IndexPolyMain(argc, argv)
#endif              /* STANDALONE */
    int             argc;
    char          **argv;
{
    IndexPoly sIPoly;
    IndexPoly       cpIPoly;

    switch (argc) {
    case 1:     /* receive sId */
        IndexPolyIn(0 /* stdin */ , &sIPoly);
        outputIPoly(stdout, &sIPoly);
        cpIPoly = sIPoly;
        outputIPoly(stdout, &cpIPoly);

        break;
    case 2:     /* receive sId */
        inputIndexPoly(stdin, &sIPoly);
#ifdef DEBUG
        outputIndexPoly(stderr, &sIPoly);
#endif
        IndexPolyOut(1 /* stdout */ , &sIPoly);

        break;
    }

}
```

```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <rpc/rpc.h>
#include <shastra/solid/indexPoly.h>

bool_t
xdr_IndexPolyVert(xdrs, objp)
    XDR *xdrs;
    IndexPolyVert objp;
{
    if (!xdr_vector(xdrs, (char *)objp, 3, sizeof(double), xdr_double)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_IndexPolyEdge(xdrs, objp)
    XDR *xdrs;
    IndexPolyEdge objp;
{
    if (!xdr_vector(xdrs, (char *)objp, 2, sizeof(int), xdr_int)) {
        return (FALSE);
    }
    return (TRUE);
}
```

```
bool_t
xdr_faceEdges(xdrs, objp)
    XDR *xdrs;
    faceEdges *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->faceEdges_val, (u_int *)&objp->
        faceEdges_len, ~0, sizeof(int), xdr_int)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_IndexPoly(xdrs, objp)
    XDR *xdrs;
    IndexPoly *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->vertices.vertices_val, (u_int *)&
        objp->vertices.vertices_len, ~0, sizeof(IndexPolyVert),
        xdr_IndexPolyVert)) {
        return (FALSE);
    }
    if (!xdr_array(xdrs, (char **)&objp->edgeVerts.edgeVerts_val, (u_int *)
        &objp->edgeVerts.edgeVerts_len, ~0, sizeof(IndexPolyEdge),
        xdr_IndexPolyEdge)) {
        return (FALSE);
    }
    if (!xdr_array(xdrs, (char **)&objp->faces.faces_val, (u_int *)&objp->
        faces.faces_len, ~0, sizeof(faceEdges), xdr_faceEdges)) {
        return (FALSE);
    }
    return (TRUE);
}
```

```
/*************************************************************************
    ***/
/*************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*************************************************************************
    ***/
/*************************************************************************
    ***/
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <rpc/rpc.h>
#include <ipoly/iPoly.h>
#include <shastra/solid/iSolid.h>

bool_t
xdr_polyTermD(xdrs, objp)
    XDR *xdrs;
    polyTermD *objp;
{
    if (!xdr_double(xdrs, &objp->coeff)) {
        return (FALSE);
    }
    if (!xdr_vector(xdrs, (char *)objp->expon, ISOLID_DIM, sizeof(short),
        xdr_short)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_polySpaD(xdrs, objp)
    XDR *xdrs;
    polySpaD *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->polySpaD_val, (u_int *)&objp->
        polySpaD_len, ~0, sizeof(polyTermD), xdr_polyTermD)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_hypRange(xdrs, objp)
    XDR *xdrs;
    hypRange objp;
{
    if (!xdr_vector(xdrs, (char *)objp, ISOLID_DIMR, sizeof(double),
        xdr_double)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_simpVertD(xdrs, objp)
    XDR *xdrs;
    simpVertD objp;
{
    if (!xdr_vector(xdrs, (char *)objp, ISOLID_DIM, sizeof(double),
        xdr_double)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_bernMixedD(xdrs, objp)
    XDR *xdrs;
    bernMixedD *objp;
{
    if (!xdr_short(xdrs, &objp->degree)) {
        return (FALSE);
    }
    if (!xdr_vector(xdrs, (char *)objp->verts, ISOLID_DIMH, sizeof
        (simpVertD), xdr_simpVertD)) {
        return (FALSE);
    }
    if (!xdr_vector(xdrs, (char *)objp->degrees, ISOLID_DIM, sizeof(short),
        xdr_short)) {
        return (FALSE);
    }
    if (!xdr_vector(xdrs, (char *)objp->hyper, ISOLID_DIM, sizeof(hypRange)
        , xdr_hypRange)) {
        return (FALSE);
    }
    if (!xdr_array(xdrs, (char **)&objp->coeffs.coeffs_val, (u_int *)&objp-
        >coeffs.coeffs_len, ~0, sizeof(double), xdr_double)) {
```

```
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_bsKnots(xdrs, objp)
    XDR *xdrs;
    bsKnots *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->bsKnots_val, (u_int *)&objp->
        bsKnots_len, ~0, sizeof(double), xdr_double)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_bSplineD(xdrs, objp)
    XDR *xdrs;
    bSplineD *objp;
{
    if (!xdr_vector(xdrs, (char *)objp->degrees, ISOLID_DIM, sizeof(short),
        xdr_short)) {
        return (FALSE);
    }
    if (!xdr_vector(xdrs, (char *)objp->knots, ISOLID_DIM, sizeof(bsKnots),
        xdr_bsKnots)) {
        return (FALSE);
    }
    if (!xdr_array(xdrs, (char **)&objp->coeffs.coeffs_val, (u_int *)&objp-
        >coeffs.coeffs_len, ~0, sizeof(double), xdr_double)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_polyEqn(xdrs, objp)
    XDR *xdrs;
    polyEqn *objp;
{
    if (!xdr_polySpaD(xdrs, objp)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_polyEqnP(xdrs, objp)
    XDR *xdrs;
    polyEqnP *objp;
{
        if (!xdr_pointer(xdrs, (char **)objp, sizeof(polyEqn), xdr_polyEqn)) {
            return (FALSE);
        }
        return (TRUE);
    }

    bool_t
    xdr_bernEqn(xdrs, objp)
        XDR *xdrs;
        bernEqn *objp;
    {
        if (!xdr_bernMixedD(xdrs, objp)) {
            return (FALSE);
        }
        return (TRUE);
    }

    bool_t
    xdr_bernEqnP(xdrs, objp)
        XDR *xdrs;
        bernEqnP *objp;
    {
        if (!xdr_pointer(xdrs, (char **)objp, sizeof(bernEqn), xdr_bernEqn)) {
            return (FALSE);
        }
        return (TRUE);
    }

    bool_t
    xdr_bSplineEqn(xdrs, objp)
        XDR *xdrs;
        bSplineEqn *objp;
    {
        if (!xdr_bSplineD(xdrs, objp)) {
            return (FALSE);
        }
        return (TRUE);
    }

    bool_t
    xdr_bSplineEqnP(xdrs, objp)
        XDR *xdrs;
        bSplineEqnP *objp;
    {
        if (!xdr_pointer(xdrs, (char **)objp, sizeof(bSplineEqn),
            xdr_bSplineEqn)) {
            return (FALSE);
        }
        return (TRUE);
    }

    bool_t
    xdr_eqnType(xdrs, objp)
    XDR *xdrs;
    eqnType *objp;
{
    if (!xdr_enum(xdrs, (enum_t *)objp)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_solBernP(xdrs, objp)
    XDR *xdrs;
    solBernP *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(struct solBern),
        xdr_solBern)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_solBern(xdrs, objp)
    XDR *xdrs;
    solBern *objp;
{
    if (!xdr_eqnType(xdrs, &objp->type)) {
        return (FALSE);
    }
    switch (objp->type) {
    case eqnIMPLI:
        if (!xdr_array(xdrs, (char **)&objp->solBern_u.implicit.
            implicit_val, (u_int *)&objp->solBern_u.implicit.implicit_len,
            ~0, sizeof(bernEqnP), xdr_bernEqnP)) {
            return (FALSE);
        }
        break;
    case eqnRATION:
        if (!xdr_array(xdrs, (char **)&objp->solBern_u.rational.
            rational_val, (u_int *)&objp->solBern_u.rational.rational_len,
            ~0, sizeof(bernEqnP), xdr_bernEqnP)) {
            return (FALSE);
        }
        break;
    case eqnPARAM:
        if (!xdr_array(xdrs, (char **)&objp->solBern_u.param.param_val,
            (u_int *)&objp->solBern_u.param.param_len, ~0, sizeof(bernEqnP)
            , xdr_bernEqnP)) {
            return (FALSE);
        }
        break;
    case eqnRATPAR:
        if (!xdr_array(xdrs, (char **)&objp->solBern_u.ratpar.ratpar_val,
            (u_int *)&objp->solBern_u.ratpar.ratpar_len, ~0, sizeof
            (bernEqnP), xdr_bernEqnP)) {
            return (FALSE);
        }
        break;
    case eqnPATCH:
        if (!xdr_array(xdrs, (char **)&objp->solBern_u.patches.patches_val,
            (u_int *)&objp->solBern_u.patches.patches_len, ~0, sizeof
            (solBernP), xdr_solBernP)) {
            return (FALSE);
        }
        break;
    }
    return (TRUE);
}

bool_t
xdr_solPolyP(xdrs, objp)
    XDR *xdrs;
    solPolyP *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(struct solPoly),
        xdr_solPoly)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_solPoly(xdrs, objp)
    XDR *xdrs;
    solPoly *objp;
{
    if (!xdr_eqnType(xdrs, &objp->type)) {
        return (FALSE);
    }
    switch (objp->type) {
    case eqnIMPLI:
        if (!xdr_array(xdrs, (char **)&objp->solPoly_u.implicit.
            implicit_val, (u_int *)&objp->solPoly_u.implicit.implicit_len,
            ~0, sizeof(polyEqnP), xdr_polyEqnP)) {
            return (FALSE);
        }
        break;
    case eqnRATION:
        if (!xdr_array(xdrs, (char **)&objp->solPoly_u.rational.
            rational_val, (u_int *)&objp->solPoly_u.rational.rational_len,
            ~0, sizeof(polyEqnP), xdr_polyEqnP)) {
            return (FALSE);
        }
        break;
    case eqnPARAM:
        if (!xdr_array(xdrs, (char **)&objp->solPoly_u.param.param_val,
            (u_int *)&objp->solPoly_u.param.param_len, ~0, sizeof(polyEqnP)
            , xdr_polyEqnP)) {
            return (FALSE);
        }
        break;
    case eqnRATPAR:
        if (!xdr_array(xdrs, (char **)&objp->solPoly_u.ratpar.ratpar_val,
            (u_int *)&objp->solPoly_u.ratpar.ratpar_len, ~0, sizeof
            (polyEqnP), xdr_polyEqnP)) {
            return (FALSE);
        }
        break;
    case eqnPATCH:
        if (!xdr_array(xdrs, (char **)&objp->solPoly_u.patches.patches_val,
            (u_int *)&objp->solPoly_u.patches.patches_len, ~0, sizeof
            (solPolyP), xdr_solPolyP)) {
            return (FALSE);
        }
        break;
    }
    return (TRUE);
}

bool_t
xdr_solBSplineP(xdrs, objp)
    XDR *xdrs;
    solBSplineP *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(struct solBSpline),
        xdr_solBSpline)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_solBSpline(xdrs, objp)
    XDR *xdrs;
    solBSpline *objp;
{
    if (!xdr_eqnType(xdrs, &objp->type)) {
        return (FALSE);
    }
    switch (objp->type) {
    case eqnIMPLI:
        if (!xdr_array(xdrs, (char **)&objp->solBSpline_u.implicit.
            implicit_val, (u_int *)&objp->solBSpline_u.implicit.
            implicit_len, ~0, sizeof(bSplineEqnP), xdr_bSplineEqnP)) {
            return (FALSE);
        }
        break;
    case eqnRATION:
        if (!xdr_array(xdrs, (char **)&objp->solBSpline_u.rational.
            rational_val, (u_int *)&objp->solBSpline_u.rational.
            rational_len, ~0, sizeof(bSplineEqnP), xdr_bSplineEqnP)) {
            return (FALSE);
        }
        break;
    case eqnPARAM:
        if (!xdr_array(xdrs, (char **)&objp->solBSpline_u.param.param_val,
            (u_int *)&objp->solBSpline_u.param.param_len, ~0, sizeof
            (bSplineEqnP), xdr_bSplineEqnP)) {
            return (FALSE);
        }
        break;
    case eqnRATPAR:
        if (!xdr_array(xdrs, (char **)&objp->solBSpline_u.ratpar.ratpar_val
            , (u_int *)&objp->solBSpline_u.ratpar.ratpar_len, ~0, sizeof
            (bSplineEqnP), xdr_bSplineEqnP)) {
            return (FALSE);
        }
        break;
    case eqnPATCH:
        if (!xdr_array(xdrs, (char **)&objp->solBSpline_u.patches.
            patches_val, (u_int *)&objp->solBSpline_u.patches.patches_len,
            ~0, sizeof(solBSplineP), xdr_solBSplineP)) {
            return (FALSE);
        }
        break;
    }
    return (TRUE);
}

bool_t
xdr_eqnBasis(xdrs, objp)
    XDR *xdrs;
    eqnBasis *objp;
{
    if (!xdr_enum(xdrs, (enum_t *)objp)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_solEqn(xdrs, objp)
    XDR *xdrs;
    solEqn *objp;
{
    if (!xdr_eqnBasis(xdrs, &objp->type)) {
        return (FALSE);
    }
    switch (objp->type) {
    case eqnPOLY:
        if (!xdr_solPoly(xdrs, &objp->solEqn_u.sPolyEqn)) {
            return (FALSE);
            }
            break;
        case eqnBERN:
            if (!xdr_solBern(xdrs, &objp->solEqn_u.sBernEqn)) {
                return (FALSE);
            }
            break;
        case eqnSPLINE:
            if (!xdr_solBSpline(xdrs, &objp->solEqn_u.sBSplineEqn)) {
                return (FALSE);
            }
            break;
        }
        return (TRUE);
    }

    bool_t
    xdr_iSolEqn(xdrs, objp)
        XDR *xdrs;
        iSolEqn *objp;
    {
        if (!xdr_array(xdrs, (char **)&objp->iSolEqn_val, (u_int *)&objp->
            iSolEqn_len, ~0, sizeof(u_int), xdr_u_int)) {
            return (FALSE);
        }
        return (TRUE);
    }

    bool_t
    xdr_iSolCycle(xdrs, objp)
        XDR *xdrs;
        iSolCycle *objp;
    {
        if (!xdr_array(xdrs, (char **)&objp->iSolCycle_val, (u_int *)&objp->
            iSolCycle_len, ~0, sizeof(u_int), xdr_u_int)) {
            return (FALSE);
        }
        return (TRUE);
    }

    bool_t
    xdr_iSolFace(xdrs, objp)
        XDR *xdrs;
        iSolFace *objp;
    {
        if (!xdr_array(xdrs, (char **)&objp->iSolFace_val, (u_int *)&objp->
            iSolFace_len, ~0, sizeof(u_int), xdr_u_int)) {
            return (FALSE);
        }
        return (TRUE);
    }

    bool_t
```

```
xdr_iSolVert(xdrs, objp)
    XDR *xdrs;
    iSolVert *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->iSolVert_val, (u_int *)&objp->
        iSolVert_len, ~0, sizeof(u_int), xdr_u_int)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_iSolEdge(xdrs, objp)
    XDR *xdrs;
    iSolEdge *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->iSolEdge_val, (u_int *)&objp->
        iSolEdge_len, ~0, sizeof(u_int), xdr_u_int)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_iSolidVerts(xdrs, objp)
    XDR *xdrs;
    iSolidVerts *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->vMarks.vMarks_val, (u_int *)&objp-
        >vMarks.vMarks_len, ~0, sizeof(u_long), xdr_u_long)) {
        return (FALSE);
    }
    if (!xdr_array(xdrs, (char **)&objp->vFaces.vFaces_val, (u_int *)&objp-
        >vFaces.vFaces_len, ~0, sizeof(iSolFace), xdr_iSolFace)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_iSolidEdges(xdrs, objp)
    XDR *xdrs;
    iSolidEdges *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->eMarks.eMarks_val, (u_int *)&objp-
        >eMarks.eMarks_len, ~0, sizeof(u_long), xdr_u_long)) {
        return (FALSE);
    }
    if (!xdr_array(xdrs, (char **)&objp->eEqns.eEqns_val, (u_int *)&objp->
        eEqns.eEqns_len, ~0, sizeof(u_int), xdr_u_int)) {
        return (FALSE);
    }
    if (!xdr_array(xdrs, (char **)&objp->eFaces.eFaces_val, (u_int *)&objp-
        >eFaces.eFaces_len, ~0, sizeof(iSolFace), xdr_iSolFace)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_iSolidCycles(xdrs, objp)
    XDR *xdrs;
    iSolidCycles *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->cMarks.cMarks_val, (u_int *)&objp-
        >cMarks.cMarks_len, ~0, sizeof(u_long), xdr_u_long)) {
        return (FALSE);
    }
    if (!xdr_array(xdrs, (char **)&objp->cFaces.cFaces_val, (u_int *)&objp-
        >cFaces.cFaces_len, ~0, sizeof(u_int), xdr_u_int)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_iSolidFaces(xdrs, objp)
    XDR *xdrs;
    iSolidFaces *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->fMarks.fMarks_val, (u_int *)&objp-
        >fMarks.fMarks_len, ~0, sizeof(u_long), xdr_u_long)) {
        return (FALSE);
    }
    if (!xdr_array(xdrs, (char **)&objp->fCycles.fCycles_val, (u_int *)&
        objp->fCycles.fCycles_len, ~0, sizeof(iSolCycle), xdr_iSolCycle)) {
        return (FALSE);
    }
    if (!xdr_array(xdrs, (char **)&objp->fVerts.fVerts_val, (u_int *)&objp-
        >fVerts.fVerts_len, ~0, sizeof(iSolVert), xdr_iSolVert)) {
        return (FALSE);
    }
    if (!xdr_array(xdrs, (char **)&objp->fEdges.fEdges_val, (u_int *)&objp-
        >fEdges.fEdges_len, ~0, sizeof(iSolEdge), xdr_iSolEdge)) {
        return (FALSE);
    }
    if (!xdr_array(xdrs, (char **)&objp->fEqns.fEqns_val, (u_int *)&objp->
        fEqns.fEqns_len, ~0, sizeof(iSolEqn), xdr_iSolEqn)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_iSolidEqns(xdrs, objp)
    XDR *xdrs;
```

```
	iSolidEqns *objp;
{
	if (!xdr_array(xdrs, (char **)&objp->sEqns.sEqns_val, (u_int *)&objp->
	    sEqns.sEqns_len, ~0, sizeof(solEqn), xdr_solEqn)) {
		return (FALSE);
	}
	return (TRUE);
}

bool_t
xdr_iSolid(xdrs, objp)
	XDR *xdrs;
	iSolid *objp;
{
	if (!xdr_iPoly(xdrs, &objp->graph)) {
		return (FALSE);
	}
	if (!xdr_iSolidVerts(xdrs, &objp->verts)) {
		return (FALSE);
	}
	if (!xdr_iSolidEdges(xdrs, &objp->edges)) {
		return (FALSE);
	}
	if (!xdr_iSolidCycles(xdrs, &objp->cycles)) {
		return (FALSE);
	}
	if (!xdr_iSolidFaces(xdrs, &objp->faces)) {
		return (FALSE);
	}
	if (!xdr_iSolidEqns(xdrs, &objp->eqns)) {
		return (FALSE);
	}
	return (TRUE);
}

bool_t
xdr_iSolid_P(xdrs, objp)
	XDR *xdrs;
	iSolid_P *objp;
{
	if (!xdr_pointer(xdrs, (char **)objp, sizeof(iSolid), xdr_iSolid)) {
		return (FALSE);
	}
	return (TRUE);
}

bool_t
xdr_iSolids(xdrs, objp)
	XDR *xdrs;
	iSolids *objp;
{
	if (!xdr_array(xdrs, (char **)&objp->iSolids_val, (u_int *)&objp->
	    iSolids_len, ~0, sizeof(iSolid), xdr_iSolid)) {
            return (FALSE);
        }
        return (TRUE);
    }

    bool_t
    xdr_iSolids_P(xdrs, objp)
        XDR *xdrs;
        iSolids_P *objp;
    {
        if (!xdr_pointer(xdrs, (char **)objp, sizeof(iSolids), xdr_iSolids)) {
            return (FALSE);
        }
        return (TRUE);
    }

    bool_t
    xdr_iSolidObj(xdrs, objp)
        XDR *xdrs;
        iSolidObj *objp;
    {
        if (!xdr_vector(xdrs, (char *)objp->sbName, ISOLID_NMLEN, sizeof(char),
            xdr_char)) {
            return (FALSE);
        }
        if (!xdr_u_long(xdrs, &objp->lIdTag)) {
            return (FALSE);
        }
        if (!xdr_u_long(xdrs, &objp->lSIdTag)) {
            return (FALSE);
        }
        if (!xdr_u_long(xdrs, &objp->lPerms)) {
            return (FALSE);
        }
        if (!xdr_u_long(xdrs, &objp->lType)) {
            return (FALSE);
        }
        if (!xdr_u_long(xdrs, &objp->lMode)) {
            return (FALSE);
        }
        if (!xdr_pointer(xdrs, (char **)&objp->pISolid, sizeof(iSolid),
            xdr_iSolid)) {
            return (FALSE);
        }
        return (TRUE);
    }

    bool_t
    xdr_iSolidObj_P(xdrs, objp)
        XDR *xdrs;
        iSolidObj_P *objp;
    {
        if (!xdr_pointer(xdrs, (char **)objp, sizeof(iSolidObj), xdr_iSolidObj)
            ) {
            return (FALSE);
        }
        return (TRUE);
    }
```

```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**********************************************************************/
/*
 * readSolid.c - input functions for solid at the network interface
 *
 * readString()
 *
 * readIndex() readAdjItem() readEqnItem()
 *
 * readVertex() readDEdge() readEdge() readCycle() readFace() readSolid()
 *
 */
/*********************************************************************/

#include <stdio.h>
#include <ctype.h>
#include <malloc.h>

#include <shastra/shilp.h>
#include <poly/poly.h>
#include <poly/polymath.h>
#include <shastra/solid/datadefs.h>
#include <shastra/solid/edgetypes.h>
#include <shastra/solid/eqntypes.h>
#include <shastra/solid/bern.h>

#include <shastra/draw/solid.h>

#include <shastra/network/server.h>
#include <shastra/solid/readSolid.h>
```

```c
char    *stdVars[3] = {"X", "Y", "Z"};

#define DEBUG 0

/*
 * readIndex(fdSocket, iptr ) - read an index into iptr
 *
 * Input should be of the form: solid# object index#
 *
 * where solid# and index# are integers, and object = V,E,F,D, or C
 */
void
readIndex(fdSocket, iptr)
     int                fdSocket;
     Index_Ptr          iptr;
{
  char            c;
  char           *sbIn;

  sbIn = readString(fdSocket);
  sscanf(sbIn, "%d %c %d", &iptr->solid, &c, &iptr->index);
  free(sbIn);
#if DEBUG
  printf("readIndex: %d %c %d\n", iptr->solid, c, iptr->index);
#endif

  switch (c) {
  case 'V':
    iptr->object = VERTEX;
    break;
  case 'E':
    iptr->object = EDGE;
    break;
  case 'F':
    iptr->object = FACE;
    break;
  case 'D':
    iptr->object = DEDGE;
    break;
  case 'C':
    iptr->object = CYCLE;
    break;
  default:
    fprintf(stderr, "Unexpected type \"%c\" in readIndex\n", c);
    break;
  }
}

/***********************************************************************/
/*
 * readAdjItem( fdSocket,aptr ) - read an adjacency into item pointer
 *
 * Input should be of the form Face_Index DEIn_Index DEOut_Index
 *
 */
/********************************************************************/
void
readAdjItem(fdSocket, aptr)
     int             fdSocket;
     AdjList_Ptr     aptr;
{
  readIndex(fdSocket, &aptr->face);
  readIndex(fdSocket, &aptr->dEIn);
  readIndex(fdSocket, &aptr->dEOut);
}

/********************************************************************/
/*
 * readEquation(fdSocket ) - read an equation , create it and return it
 */
/********************************************************************/
Poly
readEquation(fdSocket)
     int             fdSocket;
{
  char           *sbIn;
  Poly eQN;

  eQN = Parse((sbIn = readString(fdSocket)));
  free(sbIn);
  ConformPolyToVars(3, stdVars, eQN);
  return eQN;
}

/********************************************************************/
/*
 * readEqnItem(fdSocket ) - read an equation item, create it and return it
 */
/********************************************************************/
EQNList_Ptr
readEqnItem(fdSocket)
     int             fdSocket;
{
  EQNList_Ptr     New_Eqn = createEqnItem();

  New_Eqn->eQN = readEquation(fdSocket);

  return (New_Eqn);
}

/********************************************************************/
/*
 * readBernPar( fdSocket) - read bernstein-parametric eqn, return pointer
 *
 * Input should be of the form degree points...
 *
 */
/**********************************************************************/
BernPar_Ptr
readBernPar(fdSocket)
     int             fdSocket;
{
  int             degree;
  int             i;
  BernPar_Ptr     eqn;
  char           *sbIn;

  sbIn = readString(fdSocket);
  sscanf(sbIn, "%d", &degree);
  free(sbIn);

  /*
   * printf("found bernstein par eqn of degree %d\n", degree);
   */
  if (degree <= 0) {
    return NULL;
  }
  eqn = (BernPar_Ptr) malloc(sizeof(BernPar));
  eqn->degree = degree;
  eqn->coeffs = (double (*)[3])
    createMem(3 * (1 + degree) * sizeof(double));

  for (i = 0; i <= degree; i++) {
    sbIn = readString(fdSocket);
    sscanf(sbIn, "%lf %lf %lf",
       &(eqn->coeffs[i][0]),
       &(eqn->coeffs[i][1]),
       &(eqn->coeffs[i][2]));
    free(sbIn);
    /*
     * printf("read coeff  %f %f %f\n", (eqn->coeffs[i][0]),
     * (eqn->coeffs[i][1]), (eqn->coeffs[i][2]));
     */
  }
  return eqn;
}
/**********************************************************************/
/*
 * readBernParQuad( fdSocket) - read bernstein-parametric eqn, return
     pointer
 *
 * Input should be of the form degree points...
 *
 */
/**********************************************************************/
BernParQuad_Ptr
readBernParQuad(fdSocket)
     int             fdSocket;
```

```
{
  int             degree;
  int             i;
  BernParQuad_Ptr eqn;
  char           *sbIn;
  sbIn = readString(fdSocket);
  sscanf(sbIn, "%d", &degree);
  free(sbIn);

  /*
   * printf("found bernstein quad eqn of degree %d\n", degree);
   */
  if (degree <= 0) {
    return NULL;
  }
  eqn = (BernParQuad_Ptr) malloc(sizeof(BernParQuad));
  eqn->degree = degree;
  eqn->coeff1 = (double (*)[3])
    createMem(3 * (1 + degree) * sizeof(double));
  eqn->coeff2 = (double (*)[3])
    createMem(3 * (1 + degree) * sizeof(double));

  for (i = 0; i <= degree; i++) {
    sbIn = readString(fdSocket);
    sscanf(sbIn, "%lf %lf %lf",
       &(eqn->coeff1[i][0]),
       &(eqn->coeff1[i][1]),
       &(eqn->coeff1[i][2]));
    free(sbIn);
    /*
     * printf("read coeff  %f %f %f\n", (eqn->coeff1[i][0]),
     * (eqn->coeff1[i][1]), (eqn->coeff1[i][2]));
     */
  }
  for (i = 0; i <= degree; i++) {
    sbIn = readString(fdSocket);
    sscanf(sbIn, "%lf %lf %lf",
       &(eqn->coeff2[i][0]),
       &(eqn->coeff2[i][1]),
       &(eqn->coeff2[i][2]));
    free(sbIn);
    /*
     * printf("read coeff  %f %f %f\n", (eqn->coeff2[i][0]),
     * (eqn->coeff2[i][1]), (eqn->coeff2[i][2]));
     */

  }
  return eqn;
}
/*********************************************************************/
/*
 * readBernTensor( fdSocket) - read bernstein-parametric eqn, return
     pointer
 *
 * Input should be of the form degree points...
 *
 */
/********************************************************************/
BernTensor_Ptr
readBernTensor(fdSocket)
     int             fdSocket;
{
  int             degree;
  int             i;
  BernTensor_Ptr  eqn;
  char           *sbIn;
  sbIn = readString(fdSocket);
  sscanf(sbIn, "%d", &degree);
  free(sbIn);

  /*
   * printf("found bernstein tensor eqn of degree %d\n", degree);
   */
  if (degree <= 0) {
    return NULL;
  }
  eqn = (BernTensor_Ptr) malloc(sizeof(BernTensor));
  eqn->degree = degree;
  eqn->coeff1 = (double (*)[3])
    createMem(3 * (1 + degree) * sizeof(double));
  eqn->coeff2 = (double (*)[3])
    createMem(3 * (1 + degree) * sizeof(double));

  for (i = 0; i <= degree; i++) {
    sbIn = readString(fdSocket);
    sscanf(sbIn, "%lf %lf %lf",
       &(eqn->coeff1[i][0]),
       &(eqn->coeff1[i][1]),
       &(eqn->coeff1[i][2]));
    free(sbIn);
    /*
     * printf("read coeff  %f %f %f\n", (eqn->coeff1[i][0]),
     * (eqn->coeff1[i][1]), (eqn->coeff1[i][2]));
     */
  }
  for (i = 0; i <= degree; i++) {
    sbIn = readString(fdSocket);
    sscanf(sbIn, "%lf %lf %lf",
       &(eqn->coeff2[i][0]),
       &(eqn->coeff2[i][1]),
       &(eqn->coeff2[i][2]));
    free(sbIn);
    /*
     * printf("read coeff  %f %f %f\n", (eqn->coeff2[i][0]),
     * (eqn->coeff2[i][1]), (eqn->coeff2[i][2]));
     */
  }
  sbIn = readString(fdSocket);
  sscanf(sbIn, "%lf %lf %lf",
     &(eqn->tangent[0]),
     &(eqn->tangent[1]),
     &(eqn->tangent[2]));
  free(sbIn);
  /*
   * printf("read tangent  %f %f %f\n", (eqn->tangent[0]),
   * (eqn->tangent[1]), (eqn->tangent[2]));
   */

  return eqn;
}
/*********************************************************************/
/*
 * readVertex(fdSocket) - read in and create a single vertex return a
     pointer
 * to the vertex
 *
 * Input should be (assume preceeding "V" has been eaten): xval yval zval
 * #adjacencies adj1 adj2 ...
 *
 */
/*********************************************************************/
Vertex_Ptr
readVertex(fdSocket)
     int                fdSocket;
{
  Vertex_Ptr      New_Vertex = createVertex();
  AdjList_Ptr     last_adj;
  int             i, num_adj;
  double          a, b, c;
  char           *sbIn;

  /* read in the point value */
  sbIn = readString(fdSocket);
  sscanf(sbIn, "%lf %lf %lf",
     &(New_Vertex->point[0]),
     &(New_Vertex->point[1]),
     &(New_Vertex->point[2]));
  free(sbIn);

  /* read adjacencies */
  sbIn = readString(fdSocket);
  sscanf(sbIn, "%d", &num_adj);
  free(sbIn);

  /*
   * for (i = 0; i < num_adj; i++) { last_adj =
   * New_Vertex->adjacencies; New_Vertex->adjacencies =
   * createAdjItem(); New_Vertex->adjacencies->next = last_adj;
   * readAdjItem(fdSocket, New_Vertex->adjacencies); }
```

```
     */
    for (i = 0; i < num_adj; i++) {
      if (i == 0) {
        last_adj = New_Vertex->adjacencies = createAdjItem();
        readAdjItem(fdSocket, last_adj);
      } else {
        last_adj->next = createAdjItem();
        readAdjItem(fdSocket, last_adj->next);
        last_adj = last_adj->next;
      }
    }
    return (New_Vertex);
  }

  /*********************************************************************/
  /*
   * readDEdge(fdSocket) - read in and create a new directed edge
   *
   * Input should be (assume D already eaten up) cycle_index rightorientation
       (int,
   * 0 or 1) edge_index next_de_index
   */
  /*********************************************************************/
  DEdge_Ptr
  readDEdge(fdSocket)
       int              fdSocket;
  {
    DEdge_Ptr       New_DEdge = createDEdge();
    char           *sbIn;

    readIndex(fdSocket, &New_DEdge->cycle);
    sbIn = readString(fdSocket);
    sscanf(sbIn, "%d", &New_DEdge->rightOrientation);
    free(sbIn);
    readIndex(fdSocket, &New_DEdge->edge);
    readIndex(fdSocket, &New_DEdge->nextDE);

    return (New_DEdge);
  }

  /*********************************************************************/
  /*
   * readEdge(fdSocket) - read in and create an edge return a pointer to the
   * edge
   *
   * Input should be of the form (assume E eaten up):
   *
   * Name(string) V1_index V2_index Type ("LINEAR" or "BERNSTEIN_PARAMETRIC"
       or
   * "UNKNOWN") tan12_x tan12_y tan12_z tan21_x tan21_y tan21_z #of dedges
   * DirectedEdge_index1 DirectedEdge_index2 ... AUX_EQN or NO_AUX_EQN aux
       eqn,
   * as appropriate EQNS or NO_EQNS degree bernstein coeffs, as appropriate
```

```
     xi
 * yi zi
 *
 */
/*********************************************************************/
Edge_Ptr
readEdge(fdSocket)
     int              fdSocket;
{
  Edge_Ptr        New_Edge = createEdge();
  DEList_Ptr      last_de;
  int             i, num_des, degree;
  char           *sbIn;
  BernPar_Ptr     beqn;

  /* read edge name */
  sbIn = readString(fdSocket);
  sscanf(sbIn, "%19s", New_Edge->name);
  New_Edge->name[19] = '\0';
  free(sbIn);

  /* read vertex1 & vertex2 indices */
  readIndex(fdSocket, &New_Edge->vertex1);
  readIndex(fdSocket, &New_Edge->vertex2);

  /* read edge type */
  if (strncmp((sbIn = readString(fdSocket)), "LINEAR", strlen("LINEAR")) ==
      0)
    New_Edge->type = LINEAR;
  else if (strncmp(sbIn, "BERNSTEIN-TENSOR",
           strlen("BERNSTEIN-TENSOR")) == 0)
    New_Edge->type = BERNSTEIN_TENSOR_EDGE;
  else if (strncmp(sbIn, "BERNSTEIN-PARAMETRIC",
           strlen("BERNSTEIN-PARAMETRIC")) == 0)
    New_Edge->type = BERNSTEIN_PARAMETRIC;
  else if (strncmp(sbIn, "UNKNOWN", strlen("UNKNOWN")) == 0)
    New_Edge->type = UNKNOWN;
  else {
    fprintf(stderr, "Unknown edge type in readEdge -- %s\n", sbIn);
  }

  /* read tangents */
  sbIn = readString(fdSocket);
  sscanf(sbIn, "%lf %lf %lf", &New_Edge->tan12[0],
     &New_Edge->tan12[1], &New_Edge->tan12[2]);
  free(sbIn);
  sbIn = readString(fdSocket);
  sscanf(sbIn, "%lf %lf %lf", &New_Edge->tan21[0],
     &New_Edge->tan21[1], &New_Edge->tan21[2]);
  free(sbIn);

  /* read directed edges */
  sbIn = readString(fdSocket);
```

```
    sscanf(sbIn, "%d", &num_des);
    free(sbIn);
    /*
     * for (i = 0; i < num_des; i++) { last_de = New_Edge->dEdges;
     * New_Edge->dEdges = createDEdgeItem(); readIndex(fdSocket,
     * &New_Edge->dEdges->dEdge); New_Edge->dEdges->next = last_de; }
     */
    for (i = 0; i < num_des; i++) {
      if (i == 0) {
        last_de = New_Edge->dEdges = createDEdgeItem();
        readIndex(fdSocket, &last_de->dEdge);
      } else {
        last_de->next = createDEdgeItem();
        readIndex(fdSocket, &last_de->next->dEdge);
        last_de = last_de->next;
      }
    }

    /* read aux eqn */
    if (strncmp((sbIn = readString(fdSocket)),
            "AUX_EQN", strlen("AUX_EQN")) == 0) {
      free(sbIn);
      if (strncmp((sbIn = readString(fdSocket)),
          "IMPLICIT", strlen("IMPLICIT")) == 0) {
        free(sbIn);
       New_Edge->aux_Eqn = Parse((sbIn = readString(fdSocket)));
        free(sbIn);
        ConformPolyToVars(3, stdVars, New_Edge->aux_Eqn);
      } else {
        fprintf(stderr, "Unknown Aux Equation Type - %s!\n", sbIn);
        free(sbIn);
      }
    } else {
      free(sbIn);
     New_Edge->aux_Eqn = NULL;
    }

    /* see if there is a bernstein eqn */
    if (strncmp((sbIn = readString(fdSocket)), "EQNS", strlen("EQNS")) == 0)
        {
      /* read in degree */
      free(sbIn);
      if (strncmp((sbIn = readString(fdSocket)), "BERNSTEIN-PARAMETRIC",
          strlen("BERNSTEIN-PARAMETRIC")) == 0) {
        free(sbIn);
       New_Edge->eqn = readBernPar(fdSocket);
      } else {
        fprintf(stderr, "Unknown Edge Equation Type - %s!\n", sbIn);
        free(sbIn);
      }

    } else {
      free(sbIn);
```

```
  }
  return (New_Edge);
}

/*****************************************************************/
/*
 * readCycle(fdSocket) - read in, create and return a cycle
 *
 * Input should be of the form:
 *
 * face_index dedge_index
 */
/*****************************************************************/
Cycle_Ptr
readCycle(fdSocket)
     int             fdSocket;
{
  Cycle_Ptr       New_Cycle = createCycle();

  readIndex(fdSocket, &New_Cycle->face);
  readIndex(fdSocket, &New_Cycle->dEdge);

  return (New_Cycle);
}

/*****************************************************************/
/*
 * readFace(fdSocket) - read in and create a face return a pointer to the
     new
 * face
 *
 * Input should be of the form (assume F eaten): Name (string) Equation
 * (macsyma-form equation, unless bernstein) Normal_eqn_1 (macsyma form)
 * Normal_eqn_2       " Normal_eqn_3       " #cycles cycle1 cycle2 ...
 */
/*****************************************************************/
Face_Ptr
readFace(fdSocket)
     int             fdSocket;
{
  Face_Ptr        New_Face = createFace();
  EQNList_Ptr     last_eqn, next_eqn;
  CycleList_Ptr   last_cycle;
  int             i, num_cycles;
  char           *sbIn;

  /* read name */
  sbIn = readString(fdSocket);
  sscanf(sbIn, "%19s", New_Face->name);
  New_Face->name[19] = '\0';
  free(sbIn);
```

```
    /* read equation */
    if (strncmp((sbIn = readString(fdSocket)),
            "IMPLICIT", strlen("IMPLICIT")) == 0) {
      free(sbIn);
      New_Face->equation = Parse((sbIn = readString(fdSocket)));
      free(sbIn);
      ConformPolyToVars(3, stdVars, New_Face->equation);
      New_Face->type = IMPLICIT;
    } else if (strncmp(sbIn, "BERNSTEIN_PARAMETRIC_QUAD", strlen
        ("BERNSTEIN_PARAMETRIC_QUAD")) == 0) {
      free(sbIn);
      New_Face->type = BERNSTEIN_PARAMETRIC_QUAD;
      /* read it in */
      New_Face->bernQuad = readBernParQuad(fdSocket);
    } else if (strncmp(sbIn, "BERNSTEIN_TENSOR", strlen("BERNSTEIN_TENSOR"))
        == 0) {
      free(sbIn);
      New_Face->type = BERNSTEIN_TENSOR;
      /* read it in */
      New_Face->bernTens = readBernTensor(fdSocket);
    } else {
      fprintf(stderr, "Unknown Equation Type - %s!\n", sbIn);
      free(sbIn);
    }

    /* read the (three) normal equations */
    New_Face->normal = readEqnItem(fdSocket);
    New_Face->normal->next = readEqnItem(fdSocket);
    New_Face->normal->next->next = readEqnItem(fdSocket);

    /* read in the cycles */
    sbIn = readString(fdSocket);
    sscanf(sbIn, "%d", &num_cycles);
    free(sbIn);
    /*
     * last_cycle = New_Face->cycles;
     * for (i = 0; i < num_cycles; i++) { New_Face->cycles =
     * createCycleItem(); readIndex(fdSocket, &New_Face->cycles->cycle);
     * New_Face->cycles->next = last_cycle; last_cycle =
     * New_Face->cycles; }
     */
    for (i = 0; i < num_cycles; i++) {
      if (i == 0) {
        last_cycle = New_Face->cycles = createCycleItem();
        readIndex(fdSocket, &last_cycle->cycle);
      } else {
        last_cycle->next = createCycleItem();
        readIndex(fdSocket, &last_cycle->next->cycle);
        last_cycle = last_cycle->next;
      }
    }

    return (New_Face);
```

```
}

/*******************************************************************/
/*
 * readSolid(fdSocket) - read in a solid from a file return a pointer to
     the
 * new solid
 *
 * Input should be as follows (assume the preceeding "S" has already been
     eaten
 * up):
 *
 * #vert #edges #faces #dedges #cycles vertex1 vertex2 ... edge1 edge2 ...
     face1
 * face2 ... dedge1 dedge2 ... cycle1 cycle2 ...
 *
 */
/*******************************************************************/
Solid_Ptr
readSolid(fdSocket)
     int             fdSocket;
{
  Solid_Ptr       New_Solid = createSolid();
  int             i;
  Stack_Union     object;
  int             Num_Vertices, Num_Edges, Num_Faces, Num_DEdges,
      Num_Cycles;
  char           *sbIn;

  /* check for error, or solid */
  sbIn = readString(fdSocket);

  if (strncmp(sbIn, "ERROR", strlen("ERROR")) == 0) {
    free(sbIn);
    fprintf(stderr, "%s\n", sbIn);
    return (NULL);
  } else {
    free(sbIn);
  }
  /* must be SOLID # */
    sbIn = readString(fdSocket);
  sscanf(sbIn, "%19s", New_Solid->name);
  New_Solid->name[19] = '\0';
  free(sbIn);

  /* read # of vertices,edges,faces,dedges,cycles */
    sbIn = readString(fdSocket);
  sscanf(sbIn, "%d %d %d %d %d", &Num_Vertices, &Num_Edges,
     &Num_Faces, &Num_DEdges, &Num_Cycles);
  free(sbIn);

  printf("#v %d #e %d #f %d #d %d #c %d\n", Num_Vertices,
     Num_Edges, Num_Faces, Num_DEdges, Num_Cycles);
  /* read all the solid subcomponents */
  printf("reading vertices\n");
  for (i = 0; i < Num_Vertices; i++) {
    object.vertex = readVertex(fdSocket);
    sprintf(object.vertex->name, "v%d", i);
    AddObjToSolid(&object, VERTEX, New_Solid);
  }

  printf("reading edges\n");
  for (i = 0; i < Num_Edges; i++) {
    object.edge = readEdge(fdSocket);
    /*
     * sprintf(object.edge->name,"e%d",i);
     */
    AddObjToSolid(&object, EDGE, New_Solid);
  }

  printf("reading faces\n");
  for (i = 0; i < Num_Faces; i++) {
    object.face = readFace(fdSocket);
    /*
     * sprintf(object.face->name,"f%d",i);
     */
    AddObjToSolid(&object, FACE, New_Solid);
  }

  printf("reading dedges\n");
  for (i = 0; i < Num_DEdges; i++) {
    object.dEdge = readDEdge(fdSocket);
    sprintf(object.dEdge->name, "de%d", i);
    AddObjToSolid(&object, DEDGE, New_Solid);
  }

  printf("reading cycles\n");
  for (i = 0; i < Num_Cycles; i++) {
    object.cycle = readCycle(fdSocket);
    sprintf(object.cycle->name, "c%d", i);
    AddObjToSolid(&object, CYCLE, New_Solid);
  }

  return (New_Solid);
}

solidData *
readSolidData(fdSocket)
     int             fdSocket;
{
  solidData *pSolid;
  char           *sbIn;

  pSolid = (solidData*)createMem(sizeof(solidData));
  sbIn = readString(fdSocket);
  strcpy(pSolid->sbName,sbIn);

  sbIn = readString(fdSocket);
  sscanf(sbIn, "%lu%lu%lu",
     &pSolid->lIdTag,
     &pSolid->lSIdTag,
     &pSolid->lPerms);
  free(sbIn);

  sbIn = readString(fdSocket);
  sscanf(sbIn, "%d%d%d%d",
     &pSolid->dispMode,
     &pSolid->color,
     &pSolid->shade,
     &pSolid->dispInfo);
  free(sbIn);

  pSolid->pSolid = readSolid(fdSocket);

  return pSolid;
}

/*********************************************************************/
/*
 * createSolid.c - routines related to creating structures
 *
 * createMem( size ) createEntries( size ) createStack( size )
 *
 * createAdjItem() createDEdgeItem() createEqnItem() createCycleItem()
 *
 * createVertex() createEdge() createFace() createDEdge() createCycle()
 * createSolid()
 */
/*********************************************************************/



/*********************************************************************/
/* return malloc'ed memory, unless out, then crash                   */
/*********************************************************************/
char           *
createMem(size)
     unsigned        size;
{
  char           *block;

  if (size <= 0) {
    fprintf(stderr, "createMem()->requested 0 bytes\n");
    return NULL;
  }
  block = malloc(size);
```

```c
  if (block == NULL) {
    fprintf(stderr, "FATAL ERROR -- out of memory in createMem\n");
    exit(1);
  } else {
    memset(block, 0, size);
    return (block);
  }
}

/********************************************************************/
/*
 * createEntries - create an array of Stack_Union
 */
/********************************************************************/
Stack_Union    *
createEntries(size)
     int             size;
{
  return ((Stack_Union *) createMem(sizeof(Stack_Union) * size));
}

/********************************************************************/
/* create a stack with initial size given                           */
/********************************************************************/
Stack          *
createStack(size)
     int             size;
{
  Stack          *new_stack;
  new_stack = (Stack *) createMem(sizeof(Stack));

  new_stack->index = 0;
  new_stack->size = size;
  new_stack->entries = createEntries(size);
  return (new_stack);
}

/********************************************************************/
/*
 * createAdjItem()
 */
/********************************************************************/
AdjList_Ptr
createAdjItem()
{
  return ((struct AdjList *) createMem(sizeof(struct AdjList)));
}

/********************************************************************/
/*
 * createDEdgeItem()
 */
/********************************************************************/
```

```
DEList_Ptr
createDEdgeItem()
{
  return ((struct DEList *) createMem(sizeof(struct DEList)));
}

/********************************************************************/
/*
 * createEqnItem()
 */
/********************************************************************/
EQNList_Ptr
createEqnItem()
{
  return ((struct EQNList *) createMem(sizeof(struct EQNList)));
}

/********************************************************************/
/*
 * createCycleItem()
 */
/********************************************************************/
CycleList_Ptr
createCycleItem()
{
  return ((struct CycleList *) createMem(sizeof(struct CycleList)));
}

/********************************************************************/
/*
 * createVertex
 */
/********************************************************************/
Vertex         *
createVertex()
{
  return ((Vertex *) createMem(sizeof(Vertex)));
}

/********************************************************************/
/*
 * createEdge
 */
/********************************************************************/
Edge           *
createEdge()
{
  return ((Edge *) createMem(sizeof(Edge)));
}

/********************************************************************/
/*
 * createFace
 */
/**********************************************************************/
Face          *
createFace()
{
  return ((Face *) createMem(sizeof(Face)));
}

/**********************************************************************/
/*
 * createDEdge
 */
/**********************************************************************/
DEdge         *
createDEdge()
{
  return ((DEdge *) createMem(sizeof(DEdge)));
}

/**********************************************************************/
/*
 * createCycle
 */
/**********************************************************************/
Cycle         *
createCycle()
{
  return ((Cycle *) createMem(sizeof(Cycle)));
}

/**********************************************************************/
/*
 * createSolid
 */
/**********************************************************************/
Solid         *
createSolid()
{
  Solid         *new_solid = (Solid *) createMem(sizeof(Solid));

  new_solid->vertices = createStack(INITIAL_VERTICES);
  new_solid->edges = createStack(INITIAL_EDGES);
  new_solid->faces = createStack(INITIAL_FACES);
  new_solid->dEdges = createStack(INITIAL_DEDGES);
  new_solid->cycles = createStack(INITIAL_CYCLES);
  new_solid->name[0] = '\0';

  return (new_solid);
}

/**********************************************************************/
/*
 * stack.c - routines related to stack manipulation
 *
 * ReHashStack( stack ) AddObjToStack( sObject, stack ) AddObjToSolid(
     sObject,
 * Type, Solid )
 */
/********************************************************************/

/********************************************************************/
/* ReHashStack - make the given stack bigger                        */
/********************************************************************/
ReHashStack(stack)
     Stack_Ptr       stack;
{
  int             i;
  Stack_Union    *new_entries = createEntries(2 * stack->size);

  for (i = 0; i < stack->size; i++)
    new_entries[i] = stack->entries[i];

  stack->size = 2 * stack->size;
  free(stack->entries);
  stack->entries = new_entries;
}

/********************************************************************/
/* AddObjToStack - add an object to the given stack                 */
/********************************************************************/
AddObjToStack(sObject, kind, stack)
     Stack_Union    *sObject;
     int             kind;
     Stack_Ptr       stack;
{
  switch (kind) {
  case VERTEX:
    stack->entries[stack->index++].vertex = sObject->vertex;
    break;
  case EDGE:
    stack->entries[stack->index++].edge = sObject->edge;
    break;
  case FACE:
    stack->entries[stack->index++].face = sObject->face;
    break;
  case DEDGE:
    stack->entries[stack->index++].dEdge = sObject->dEdge;
    break;
  case CYCLE:
    stack->entries[stack->index++].cycle = sObject->cycle;
    break;
  default:
    fprintf(stderr, "Attempt to AddObjToStack unknown object type #%d\n",
        kind);
    exit(1);
    break;
```

```c
  }

  if ((stack->index + 1) == stack->size)
    ReHashStack(stack);
}

/********************************************************************/
/* AddObjToSolid - add an object to the given solid                 */
/********************************************************************/
AddObjToSolid(sObject, kind, S)
     Stack_Union   *sObject;
     int             kind;
     Solid_Ptr       S;
{
  switch (kind) {
  case VERTEX:
    AddObjToStack(sObject, kind, S->vertices);
    break;
  case EDGE:
    AddObjToStack(sObject, kind, S->edges);
    break;
  case FACE:
    AddObjToStack(sObject, kind, S->faces);
    break;
  case DEDGE:
    AddObjToStack(sObject, kind, S->dEdges);
    break;
  case CYCLE:
    AddObjToStack(sObject, kind, S->cycles);
    break;
  default:
    fprintf(stderr, "Attempt to AddObjToSolid unknown object type #%d\n",
        kind);
    exit(1);
    break;
  }
}
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
#include <stdio.h>
#include <ctype.h>
#include <shastra/solid/vIndexPolyH.h>
#include <shastra/network/server.h>
#include <shastra/network/mplex.h>
#include <shastra/network/rpc.h>

#define STANDALONEnn

static char         sbOut[5120];

int
vIndexPolyOut(fd, pIPoly)
    int             fd;
vIndexPoly      *pIPoly;
{
    XDR             xdrs;
    int             retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdout /* fdopen(fd,"w") */ ;
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
        if (!xdr_vIndexPoly(&xdrs, pIPoly)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
```

```
    /*
     * xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
     */
    if (!xdr_vIndexPoly(mplexXDRSEnc(fd), pIPoly)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
vIndexPolyIn(fd, pIPoly)
    int             fd;
vIndexPoly      *pIPoly;
{
    XDR             xdrs;
    int             retVal = 0;

    vIndexPolyXDRFree(pIPoly);
#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdin /* fdopen(fd,"r") */ ;
        xdrstdio_create(&xdrs, fp, XDR_DECODE);
        if (!xdr_vIndexPoly(&xdrs, pIPoly)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);
     */
    if (!xdr_vIndexPoly(mplexXDRSDec(fd), pIPoly)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}



void
inputVIndexPoly(fp, pIPoly)
    FILE           *fp;
vIndexPoly      *pIPoly;
{
    int             i,j;

    fscanf(fp, "%u", &pIPoly->vertices.vertices_len);
    pIPoly->vertices.vertices_val =
        (vIndexPolyVert *) malloc(sizeof(vIndexPolyVert) *
                             pIPoly->vertices.vertices_len);
    for (i = 0; i < pIPoly->vertices.vertices_len; i++) {
        fscanf(fp, "%lf%lf%lf",
               &pIPoly->vertices.vertices_val[i][0],
               &pIPoly->vertices.vertices_val[i][1],
               &pIPoly->vertices.vertices_val[i][2]);
    }

    fscanf(fp, "%u", &pIPoly->faces.faces_len);
    pIPoly->faces.faces_val =
        (faceVerts *) malloc(sizeof(faceVerts) *
                             pIPoly->faces.faces_len);
    for (i = 0; i < pIPoly->faces.faces_len; i++) {
        fscanf(fp, "%u", &pIPoly->faces.faces_val[i].faceVerts_len);
        pIPoly->faces.faces_val[i].faceVerts_val =
            (int *) malloc(sizeof(int) *
                  pIPoly->faces.faces_val[i].faceVerts_len);
        for (j = 0; j < pIPoly->faces.faces_val[i].faceVerts_len; j++) {
            fscanf(fp, "%d",
                   &pIPoly->faces.faces_val[i].faceVerts_val[j]);
        }
    }
}

void
outputVIndexPoly(fp, pIPoly)
    FILE           *fp;
vIndexPoly      *pIPoly;
{
    int             i,j;

    fprintf(fp, "%u\n", pIPoly->vertices.vertices_len);
    for (i = 0; i < pIPoly->vertices.vertices_len; i++) {
        fprintf(fp, "%lf %lf %lf\n",
            pIPoly->vertices.vertices_val[i][0],
            pIPoly->vertices.vertices_val[i][1],
            pIPoly->vertices.vertices_val[i][2]);
    }

    fprintf(fp, "%u\n", pIPoly->faces.faces_len);
    for (i = 0; i < pIPoly->faces.faces_len; i++) {
        fprintf(fp, "%u\n", pIPoly->faces.faces_val[i].faceVerts_len);
        for (j = 0; j < pIPoly->faces.faces_val[i].faceVerts_len; j++) {
            fprintf(fp, "%d ",
                pIPoly->faces.faces_val[i].faceVerts_val[j]);
        }
        fprintf(fp, "\n");
    }
}



void
freeVIndexPoly(pIPoly)
vIndexPoly      *pIPoly;
```

```
{
    int                 i;

    free(pIPoly->vertices.vertices_val);

    for (i = 0; i < pIPoly->faces.faces_len; i++) {
        free(pIPoly->faces.faces_val[i].faceVerts_val);
    }
    free(pIPoly->faces.faces_val);
    memset(pIPoly, 0, sizeof(vIndexPoly));
}

vIndexPoly      *
copyVIndexPoly(pIPoly, destpIPoly)
vIndexPoly      *pIPoly;
vIndexPoly      *destpIPoly;
{
vIndexPoly      *newpIPoly;
    int                 i;

    if (pIPoly == NULL) {
        return NULL;
    }
    if (destpIPoly == NULL) {
        newpIPoly = (vIndexPoly *) malloc(sizeof(vIndexPoly));
    } else {
        newpIPoly = destpIPoly;
    }

    destpIPoly->vertices.vertices_len = pIPoly->vertices.vertices_len;
    destpIPoly->vertices.vertices_val =
        (vIndexPolyVert *) malloc(sizeof(vIndexPolyVert) *
                     pIPoly->vertices.vertices_len);
    memcpy(destpIPoly->vertices.vertices_val,
            pIPoly->vertices.vertices_val,
          sizeof(vIndexPolyVert) *
          pIPoly->vertices.vertices_len);

    destpIPoly->faces.faces_len = pIPoly->faces.faces_len;
    destpIPoly->faces.faces_val =
        (faceVerts *) malloc(sizeof(faceVerts) *
                     pIPoly->faces.faces_len);
    for (i = 0; i < pIPoly->faces.faces_len; i++) {
        destpIPoly->faces.faces_val[i].faceVerts_len =
            pIPoly->faces.faces_val[i].faceVerts_len;
        destpIPoly->faces.faces_val[i].faceVerts_val =
            (int *) malloc(sizeof(int) *
                  pIPoly->faces.faces_val[i].faceVerts_len);
        memcpy( destpIPoly->faces.faces_val[i].faceVerts_val,
            pIPoly->faces.faces_val[i].faceVerts_val,
            sizeof(int) * pIPoly->faces.faces_val[i].faceVerts_len);
    }
    return destpIPoly;
```

```
}

void
vIndexPolyXDRFree(pIPoly)
vIndexPoly      *pIPoly;
{
    xdr_free(xdr_vIndexPoly, (char *) pIPoly);
    memset(pIPoly, 0, sizeof(vIndexPoly));
}




vIndexPoly      *
inputVIndexPolyString(fd)
    int             fd;
{
vIndexPoly      *pIPoly;
    int             i,j;
    char *sbIn;

    pIPoly = (vIndexPoly*)malloc(sizeof(vIndexPoly));
    memset(pIPoly, 0,sizeof(vIndexPoly));
    sbIn = cmReceiveString(fd);
    sscanf(sbIn, "%u", &pIPoly->vertices.vertices_len);
    free(sbIn);
    pIPoly->vertices.vertices_val =
        (vIndexPolyVert *) malloc(sizeof(vIndexPolyVert) *
                     pIPoly->vertices.vertices_len);
    for (i = 0; i < pIPoly->vertices.vertices_len; i++) {
        sbIn = cmReceiveString(fd);
        sscanf(sbIn, "%lf%lf%lf",
               &pIPoly->vertices.vertices_val[i][0],
               &pIPoly->vertices.vertices_val[i][1],
               &pIPoly->vertices.vertices_val[i][2]);
        free(sbIn);
    }

    sbIn = cmReceiveString(fd);
    sscanf(sbIn, "%u", &pIPoly->faces.faces_len);
    free(sbIn);
    pIPoly->faces.faces_val =
        (faceVerts *) malloc(sizeof(faceVerts) *
                     pIPoly->faces.faces_len);
    for (i = 0; i < pIPoly->faces.faces_len; i++) {
        char *iptr;
        sbIn = cmReceiveString(fd);
        sscanf(sbIn, "%u", &pIPoly->faces.faces_val[i].faceVerts_len);
        free(sbIn);
        pIPoly->faces.faces_val[i].faceVerts_val =
            (int *) malloc(sizeof(int) *
                  pIPoly->faces.faces_val[i].faceVerts_len);
        iptr = sbIn = cmReceiveString(fd);
        for (j = 0; j < pIPoly->faces.faces_val[i].faceVerts_len; j++) {
            while((!isdigit(*iptr)) && (*iptr!='-')){
                iptr++/*skip nonnumerics*/;
            }
            sscanf(iptr, "%d",
                   &pIPoly->faces.faces_val[i].faceVerts_val[j]);
            if(*iptr == '-'){
                iptr++;
            }
            while(isdigit(*iptr))iptr++/*skip numerics*/;
        }
        free(sbIn);
    }
    return pIPoly;
}

void
outputVIndexPolyString(fd, pIPoly)
    int     fd;
vIndexPoly      *pIPoly;
{
    int             i,j;

    sprintf(sbOut, "%u\n", pIPoly->vertices.vertices_len);
    cmSendString(fd,sbOut);
    for (i = 0; i < pIPoly->vertices.vertices_len; i++) {
        sprintf(sbOut, "%lf %lf %lf\n",
            pIPoly->vertices.vertices_val[i][0],
            pIPoly->vertices.vertices_val[i][1],
            pIPoly->vertices.vertices_val[i][2]);
        cmSendString(fd,sbOut);
    }

    sprintf(sbOut, "%u\n", pIPoly->faces.faces_len);
    cmSendString(fd,sbOut);
    for (i = 0; i < pIPoly->faces.faces_len; i++) {
        char *optr;
        sprintf(sbOut, "%u\n", pIPoly->faces.faces_val[i].faceVerts_len);
        cmSendString(fd,sbOut);
        optr = sbOut;
        for (j = 0; j < pIPoly->faces.faces_val[i].faceVerts_len; j++) {
            sprintf(optr, "%d ",
                pIPoly->faces.faces_val[i].faceVerts_val[j]);
            optr += strlen(optr);
        }
        sprintf(optr, "\n");
        cmSendString(fd,sbOut);
    }
}
```

```
#ifdef STANDALONE
main(argc, argv)
#else               /* STANDALONE */
vIndexPolyMain(argc, argv)
#endif              /* STANDALONE */
    int             argc;
    char          **argv;
{
vIndexPoly sIPoly;
vIndexPoly       cpIPoly;

    switch (argc) {
    case 1:     /* receive sId */
    vIndexPolyIn(0 /* stdin */ , &sIPoly);
        outputVIndexPoly(stdout, &sIPoly);
        cpIPoly = sIPoly;
        outputVIndexPoly(stdout, &cpIPoly);

        break;
    case 2:     /* receive sId */
        inputVIndexPoly(stdin, &sIPoly);
#ifdef DEBUG
        outputVIndexPoly(stderr, &sIPoly);
#endif
    vIndexPolyOut(1 /* stdout */ , &sIPoly);

        break;
    }

}
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/*
 * write.c - output functions for the network interface
 *
 * writeString()
 *
 * writeIndex( iptr ) writeAdjItem( aptr ) writeEqn(eptr)
 *
 * writeVertex( vptr) writeDEdge(deptr) writeEdge(eptr) writeCycle(cptr)
 * writeFace(fptr) writeSolid(sptr)
 *
 */


#include <stdio.h>

#include <shastra/shilp.h>
#include <poly/poly.h>
#include <poly/polymath.h>
#include <shastra/solid/datadefs.h>
#include <shastra/solid/edgetypes.h>
#include <shastra/solid/eqntypes.h>
#include <shastra/solid/bern.h>

#include <shastra/solid/writeSolid.h>

static char         sbOut[5120];
char            *sbVarNames[] = {"X", "Y", "Z"};
int              iVarCount = 3;
/* implicit power equations will always be in x,y & z */
```

```
/*
 * writeString(fdSocket, s ) - write string
 */
void
writeString(fdSocket, s)
    int fdSocket;
    char           *s;
{
    cmSendString(fdSocket, s);
}

/*
 * writeStrings(fdSocket,n,strs) - strs n strings given n, char ** array
 */
void
writeStrings(fdSocket,number,names)
int fdSocket;
int number;
char**names;
{
    int             i;
    int len;

    sprintf( sbOut ,"%d", number);
    writeString(fdSocket,sbOut);

    if(number <= 0){
        return ;
    }
    for (i = 0; i < number; i++) {
    sprintf( sbOut ,"%s", names[i]);
    writeString(fdSocket,sbOut);
    }

    return ;

}               /* end readStrings */
/*
 * writeIndex(fdSocket, iptr ) - write an index from iptr
 */
void
writeIndex(fdSocket, iptr)
    int fdSocket;
    Index_Ptr       iptr;
{
    char            c;

    switch (iptr->object) {
    case VERTEX:
        c = 'V';
        break;
    case EDGE:
            c = 'E';
            break;
        case FACE:
            c = 'F';
            break;
        case DEDGE:
            c = 'D';
            break;
        case CYCLE:
            c = 'C';
            break;
        default:
            fprintf(stderr, "ERROR:Unexpected type %d in writeIndex\n",
                iptr->object);
            break;
        }

        sprintf(sbOut, "%d %c %d\n", iptr->solid, c, iptr->index);
        writeString(fdSocket,sbOut);
    #if DEBUG
        printf("writeIndex: %d %c %d", iptr->solid, c, iptr->index);
    #endif
    }

    /*
     * writeAdjItem( fdSocket, aptr ) -
     */
    void
    writeAdjItem(fdSocket, aptr)
        int fdSocket;
        AdjList_Ptr     aptr;
    {
        writeIndex(fdSocket, &aptr->face);
        writeIndex(fdSocket, &aptr->dEIn);
        writeIndex(fdSocket, &aptr->dEOut);
    }
    /*
     * writeEqn(fdSocket, New_Eqn) -
     */
    void
    writeEqn(int fdSocket, Poly New_Eqn)
    {
        char *sbEqn;

        sbEqn = UnParse(New_Eqn);
        sprintf(sbOut, "%s\n", sbEqn);
        writeString(fdSocket,sbOut);
    }

    /*
     * writeBernPar( fdSocket, BernPar_Ptr) - write bernstein-parametric eqn
     */
    void
```

```
writeBernPar(fdSocket, eqn)
    int fdSocket;
    BernPar_Ptr eqn;
{
int i;
            sprintf(sbOut, "%d\n", eqn->degree);
            writeString(fdSocket,sbOut);

            if(eqn->degree <= 0){
                return ;
            }
            for (i = 0; i <= eqn->degree; i++) {
                sprintf(sbOut, "%lf %lf %lf\n",
                       eqn->coeffs[i][0],
                       eqn->coeffs[i][1],
                       eqn->coeffs[i][2]);
                writeString(fdSocket,sbOut);
            }
            return ;
}
/*
 * writeBernParQuad( fdSocket, eqn) - write bernstein-parametric quad
 */
void
writeBernParQuad(fdSocket, eqn)
    int fdSocket;
    BernParQuad_Ptr eqn;
{
int i;
            sprintf(sbOut, "%d\n", eqn->degree);
            writeString(fdSocket,sbOut);

            if(eqn->degree <= 0){
                return ;
            }
            for (i = 0; i <= eqn->degree; i++) {
                sprintf(sbOut, "%lf %lf %lf\n",
                       eqn->coeff1[i][0],
                       eqn->coeff1[i][1],
                       eqn->coeff1[i][2]);
                writeString(fdSocket,sbOut);
            }
            for (i = 0; i <= eqn->degree; i++) {
                sprintf(sbOut, "%lf %lf %lf\n",
                       eqn->coeff2[i][0],
                       eqn->coeff2[i][1],
                       eqn->coeff2[i][2]);
                writeString(fdSocket,sbOut);
            }
            return ;
}
/*
 * writeBernTensor( fdSocket, eqn) - write bernstein-tensor eqn
 */
void
writeBernTensor(fdSocket, eqn)
    int fdSocket;
    BernTensor_Ptr eqn;
{
int i;
            sprintf(sbOut, "%d\n", eqn->degree);
            writeString(fdSocket,sbOut);

            if(eqn->degree <= 0){
                return ;
            }
            for (i = 0; i <= eqn->degree; i++) {
                sprintf(sbOut, "%lf %lf %lf\n",
                       eqn->coeff1[i][0],
                       eqn->coeff1[i][1],
                       eqn->coeff1[i][2]);
                writeString(fdSocket,sbOut);
            }
            for (i = 0; i <= eqn->degree; i++) {
                sprintf(sbOut, "%lf %lf %lf\n",
                       eqn->coeff2[i][0],
                       eqn->coeff2[i][1],
                       eqn->coeff2[i][2]);
                writeString(fdSocket,sbOut);
            }
            sprintf(sbOut, "%lf %lf %lf\n",
                   eqn->tangent[0],
                   eqn->tangent[1],
                   eqn->tangent[2]);
            return ;
}
/*
 * writeVertex(fdSocket) -
 */
void
writeVertex(fdSocket, New_Vertex)
    int fdSocket;
    Vertex_Ptr      New_Vertex;
{
    AdjList_Ptr     last_adj;
    int             i, num_adj;

    /* write in the point value */
    sprintf(sbOut, "%lf %lf %lf\n",
        New_Vertex->point[0],
        New_Vertex->point[1],
        New_Vertex->point[2]);
    writeString(fdSocket,sbOut);

    /* write adjacencies */
    for (num_adj = 0, last_adj = New_Vertex->adjacencies;
            last_adj != NULL;
            num_adj++, last_adj = last_adj->next) {
        }
        sprintf(sbOut, "%d\n", num_adj);
        writeString(fdSocket,sbOut);

        for (last_adj = New_Vertex->adjacencies;
             last_adj != NULL;
             last_adj = last_adj->next) {
            writeAdjItem(fdSocket, last_adj);
        }

    }

    /*
     * writeDEdge(fdSocket) -
     */
    void
    writeDEdge(fdSocket, New_DEdge)
        int fdSocket;
        DEdge_Ptr       New_DEdge;
    {

        writeIndex(fdSocket, &New_DEdge->cycle);

        sprintf(sbOut, "%d\n", New_DEdge->rightOrientation);
        writeString(fdSocket,sbOut);

        writeIndex(fdSocket, &New_DEdge->edge);
        writeIndex(fdSocket, &New_DEdge->nextDE);

    }

    /*
     * writeEdge(fdSocket) -
     *
     */
    void
    writeEdge(fdSocket, New_Edge)
        int fdSocket;
        Edge_Ptr        New_Edge;
    {
        DEList_Ptr      last_de;
        char            temp_string[80];
        int             i, num_des;

        /* write edge name */
        sprintf(sbOut, "%s\n", New_Edge->name);
        writeString(fdSocket,sbOut);

        /* write vertex1 & vertex2 indices */
        writeIndex(fdSocket, &New_Edge->vertex1);
        writeIndex(fdSocket, &New_Edge->vertex2);
    /* write edge type */
    switch (New_Edge->type) {
    case LINEAR:
        sprintf(sbOut, "%s\n", "LINEAR");
        break;
    case BERNSTEIN_PARAMETRIC:
        sprintf(sbOut, "%s\n", "BERNSTEIN-PARAMETRIC");
        break;
    case BERNSTEIN_TENSOR_EDGE:
        sprintf(sbOut, "%s\n", "BERNSTEIN-TENSOR");
        break;
    case UNKNOWN:
        sprintf(sbOut, "%s\n", "UNKNOWN");
        break;
    default:
        sprintf(sbOut, "%s\n", "ERROR_EDGE_TYPE");
        fprintf(stderr, "Unknown edge type in writeEdge\n");
        exit(1);
    }
    writeString(fdSocket,sbOut);

    /* write tangents */
    sprintf(sbOut, "%lf %lf %lf\n", New_Edge->tan12[0],
        New_Edge->tan12[1], New_Edge->tan12[2]);
    writeString(fdSocket,sbOut);

    sprintf(sbOut, "%lf %lf %lf\n", New_Edge->tan21[0],
        New_Edge->tan21[1], New_Edge->tan21[2]);
    writeString(fdSocket,sbOut);

    /* write directed edges */
    for (num_des = 0, last_de = New_Edge->dEdges;
         last_de != NULL;
         num_des++, last_de = last_de->next) {
    }
    sprintf(sbOut, "%d\n", num_des);
    writeString(fdSocket,sbOut);

    for (last_de = New_Edge->dEdges;
         last_de != NULL;
         last_de = last_de->next) {
        writeIndex(fdSocket, &last_de->dEdge);
    }

    /* write aux eqn */
    if (New_Edge->aux_Eqn != NULL) {
        sprintf(sbOut, "%s\n", "AUX_EQN");
        writeString(fdSocket,sbOut);
        sprintf(sbOut, "%s\n", "IMPLICIT");
        writeString(fdSocket,sbOut);
        writeEqn(fdSocket, New_Edge->aux_Eqn);
    } else {
            sprintf(sbOut, "%s\n", "NO_AUX_EQN");
            writeString(fdSocket,sbOut);
        }

        /* write bern eqn */
        if ((New_Edge->eqn != NULL) && ( New_Edge->eqn->degree > 0)) {
            sprintf(sbOut, "EQNS\n");
            writeString(fdSocket,sbOut);
            sprintf(sbOut, "BERNSTEIN-PARAMETRIC\n");
            writeString(fdSocket,sbOut);
            writeBernPar(fdSocket,New_Edge->eqn);
        } else {
            sprintf(sbOut, "%s\n", "NO_EQNS");
            writeString(fdSocket,sbOut);
        }

    }

    /*
     * writeCycle(fdSocket) -
     */
    void
    writeCycle(fdSocket, New_Cycle)
        int fdSocket;
        Cycle_Ptr       New_Cycle;
    {

        writeIndex(fdSocket, &New_Cycle->face);
        writeIndex(fdSocket, &New_Cycle->dEdge);
    }

    /*
     * writeFace(fdSocket, New_Face) -
     */
    void
    writeFace(fdSocket, New_Face)
        int fdSocket;
        Face_Ptr        New_Face;
    {
        EQNList_Ptr     last_eqn, next_eqn;
        CycleList_Ptr   last_cycle;
        int             i, num_cycles;
        char           *b;

        /* write name */
        sprintf(sbOut, "%s\n", New_Face->name);
        writeString(fdSocket,sbOut);

        /* write equation */
        switch (New_Face->type) {
        case IMPLICIT:
            sprintf(sbOut, "IMPLICIT\n");
            writeString(fdSocket,sbOut);
            writeEqn(fdSocket, New_Face->equation);
            break;
        case BERNSTEIN_PARAMETRIC_QUAD:
            sprintf(sbOut, "BERNSTEIN_PARAMETRIC_QUAD\n");
            writeString(fdSocket,sbOut);
            /* write it out */
            writeBernParQuad(fdSocket,New_Face->bernQuad);
            break;
        case BERNSTEIN_TENSOR:
            sprintf(sbOut, "BERNSTEIN_TENSOR\n");
            writeString(fdSocket,sbOut);
            /* write it out */
            writeBernTensor(fdSocket,New_Face->bernTens);
            break;
        default:
            break;
        }

        /* write the (three) normal equations */
        writeEqn(fdSocket, New_Face->normal->eQN);
        writeEqn(fdSocket, New_Face->normal->next->eQN);
        writeEqn(fdSocket, New_Face->normal->next->next->eQN);

        /* write in the cycles */
        for (num_cycles = 0, last_cycle = New_Face->cycles;
             last_cycle != NULL;
             num_cycles++, last_cycle = last_cycle->next) {
        }
        sprintf(sbOut, "%d\n", num_cycles);
        writeString(fdSocket,sbOut);

        for (last_cycle = New_Face->cycles;
             last_cycle != NULL;
             last_cycle = last_cycle->next) {
            writeIndex(fdSocket, &last_cycle->cycle);
        }
    }

    /*
     * writeSolid(fdSocket) -
     *
     */
    void
    writeSolid(fdSocket, New_Solid)
        int fdSocket;
        Solid_Ptr       New_Solid;
    {
        int             i;
        int             Num_Vertices, Num_Edges, Num_Faces, Num_DEdges,
                        Num_Cycles;

        if (New_Solid == NULL) {
            fprintf(stderr, "writeSolid(): Can't write NULL solid!\n");
        return;
    }
    Num_Vertices = New_Solid->vertices->index,
        Num_Edges = New_Solid->edges->index,
        Num_Faces = New_Solid->faces->index,
        Num_DEdges = New_Solid->dEdges->index,
        Num_Cycles = New_Solid->cycles->index;

    sprintf(sbOut, "SOLID %d\n", 1);
    writeString(fdSocket,sbOut);

    sprintf(sbOut, "%s\n", New_Solid->name);
    writeString(fdSocket,sbOut);

    /* write # of vertices,edges,faces,dedges,cycles */
    sprintf(sbOut, "%d %d %d %d %d\n", Num_Vertices, Num_Edges,
        Num_Faces, Num_DEdges, Num_Cycles);
    writeString(fdSocket,sbOut);

    /* write all the solid subcomponents */
    for (i = 0; i < Num_Vertices; i++) {
        writeVertex(fdSocket, New_Solid->vertices->entries[i].vertex);
    }

    for (i = 0; i < Num_Edges; i++) {
        writeEdge(fdSocket, New_Solid->edges->entries[i].edge);
    }

    for (i = 0; i < Num_Faces; i++) {
        writeFace(fdSocket, New_Solid->faces->entries[i].face);
    }

    for (i = 0; i < Num_DEdges; i++) {
        writeDEdge(fdSocket, New_Solid->dEdges->entries[i].dEdge);
    }

    for (i = 0; i < Num_Cycles; i++) {
        writeCycle(fdSocket, New_Solid->cycles->entries[i].cycle);
    }

    /*
    fflush(fdSocket);
    */
    return;
}

void
writeSolidData(fdSocket, pSolid)
    int             fdSocket;
    solidData *pSolid;
{

    sprintf( sbOut ,"%s", pSolid->sbName);
    writeString(fdSocket,sbOut);

    sprintf(sbOut, "%lu %lu %lu",
        pSolid->lIdTag,
        pSolid->lSIdTag,
        pSolid->lPerms);
    writeString(fdSocket,sbOut);

    sprintf(sbOut, "%d %d %d %d",
        pSolid->dispMode,
        pSolid->color,
        pSolid->shade,
        pSolid->dispInfo);
    writeString(fdSocket,sbOut);

    writeSolid(fdSocket, pSolid->pSolid);
    return;
}
/*********************************************************************/
/*
 * Print_Expr2Str  -- prints an expression as a list of terms
 */
/*********************************************************************/
void
Print_Expr2Str(termlist, str, fWantZeros)
    TermList        termlist;
    char           *str;
    int             fWantZeros;
{
    TermList        temp = termlist;
    int             i;
    int             fAny;
    int             fPrevTerm;

    if (temp == NULL) {
        sprintf(str, "(null)\n");
    }
    fAny = 0;
    fPrevTerm = 0;
    while (temp != NULL) {
        /* print the coefficient, and then the terms */
        if (temp->term.coeff == 0.0) {
            temp = temp->next;
            continue;
        }
        if (fPrevTerm) {
            sprintf(str, " + ");
            str += strlen(str);
        }
        /* print the coefficient */
        sprintf(str, "%10f ", temp->term.coeff);
        str += strlen(str);
        fAny = 1;
        fPrevTerm = 1;

        for (i = 0; i < iVarCount; i++) {
            if (fWantZeros || (temp->term.exponents[i] != 0)) {
                sprintf(str, " * %s^%d ", sbVarNames[i],
                    temp->term.exponents[i]);
                str += strlen(str);
            }
        }
        temp = temp->next;
    }
    if (!fAny) {
        sprintf(str, "0.0");
        str += strlen(str);
    }
}
/*********************************************************************/
/*
 * Print_Expr2File  -- prints an expression as a list of terms
 */
/*********************************************************************/
void
Print_Expr2File(file, termlist, fWantZeros)
    FILE *file;
    TermList        termlist;
    int             fWantZeros;
{
    TermList        temp = termlist;
    int             i;
    int             fAny;
    int             fPrevTerm;

    if (temp == NULL) {
        fprintf(file, "(null)\n");
    }
    fAny = 0;
    fPrevTerm = 0;
    while (temp != NULL) {
        /* print the coefficient, and then the terms */
        if (temp->term.coeff == 0.0) {
            temp = temp->next;
            continue;
        }
        if (fPrevTerm) {
            fprintf(file, " + ");
        }
        /* print the coefficient */
        fprintf(file, "%10f ", temp->term.coeff);
        fAny = 1;
        fPrevTerm = 1;

        for (i = 0; i < iVarCount; i++) {
            if (fWantZeros || (temp->term.exponents[i] != 0)) {
                fprintf(file, " * %s^%d ", sbVarNames[i],
                    temp->term.exponents[i]);
            }
        }
        temp = temp->next;
    }
    if (!fAny) {
        fprintf(file, "0.0");
    }
}
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
#include <stdio.h>

#include <shastra/shilp.h>

/*command line argument processing utility */

usage(argc,argv,argvHelp)
int argc;
char *argv[];
char *argvHelp[];
{
    int i;

    fprintf(stderr, "usage: %s [options]\n", argv[0]);
    fprintf(stderr, "  where options are:\n");
    for(i=0;arvgHelp[i]!=NULL;i++){
        fprintf(stderr, "%s\n", argvHelp[i]);
    }
}

cmdLineOpts(argc,argv)
int argc;
char *argv[];
{
int i;
    for (i = 1; i < argc; i++) {
        if (!strcmp ("-display", argv[i]) || !strcmp ("-d", argv[i])) {
            if (++i>=argc) usage ();
            display_name = argv[i];
            continue;
        }
        if (!strcmp("-help", argv[i])) {
            usage();
        }
    /*etc..*/
    usage();
    }

}
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
#include <stdio.h>
#include <sys/types.h>
#include <sys/dir.h>

#include <shastra/utils/directory.h>
#define NOT_FOUND -1
#define DEBUG
#define STANDALONEnn

int
locateNameInDir(name, dirname)
    char           *name, *dirname;
{
    DIR            *dirp;
    struct direct  *dp;
    int             len;
    int             found = 0;
    len = strlen(name);
    if ((dirp = opendir(dirname)) == NULL) {
        fprintf(stderr, "locateNameInDir()-> Couldn't open directory %s\n",
            dirname);
        return NOT_FOUND;
    }
    for (dp = readdir(dirp), found = 0; dp != NULL;
         dp = readdir(dirp), found++)
        if (dp->d_namlen == len && !strcmp(dp->d_name, name)) {
            closedir(dirp);
            return found;
        }
    closedir(dirp);
    return NOT_FOUND;

}

int
forAllFilesInDir(dirname, doit)
    char           *dirname;
    void             (*doit) ();
{
    DIR            *dirp;
    struct direct  *dp;
    if ((dirp = opendir(dirname)) == NULL) {
        fprintf(stderr, "forAllFilesInDir()-> Couldn't open dir %s\n",
            dirname);
        return NOT_FOUND;
    }
        for (dp = readdir(dirp); dp != NULL;
             dp = readdir(dirp)) {
            doit(dp->d_name, dirname);
        }
    closedir(dirp);
    return 0;

}

void
dumdoit(str, n)
    char           *str;
    int             n;
{
    printf("%s ", str);
}

#ifdef STANDALONE
main(argc, argv, envp)
    int             argc;
    char          **argv, **envp;
{
    int             found;

    if (argc != 2) {
        fprintf(stderr,"bad usage.. %s name\n", argv[0]);
        exit(1);
    };
    if (argc == 2) {
        found = locateNameInDir(argv[1], ".");
        if(found != NOT_FOUND){
        printf("Found %s in %s at %d'th position\n", argv[1], ".", found);
        }
        else{
        printf("Couldn't find %s in %s\n", argv[1], ".", found);
        }
    }

    forAllFilesInDir(".", dumdoit);

}
#endif /*STANDALONE*/
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
#include <stdio.h>

#include <shastra/shilp.h>
#include <shastra/utils/dllist.h>

extern          free();

int
dllistCheckGood(adllist)
    struct dllist    *adllist;
{
    int             baddllist = 1;

    if (adllist == NULL) {
        fprintf(stderr, "BadArgs to dllistCheckGood)\n");
        return (0);
    }
    if (adllist->head == NULL) {
        if (adllist->tail == NULL) {
            if (adllist->dllist_count != 0) {
                baddllist = 0;
            }
        } else {
            baddllist = 0;
        }
    } else {
        if (adllist->tail == NULL) {
            baddllist = 0;
        }
    }
    if (!baddllist) {
        return 0;
    } else {
        return dllistCheckCount(adllist);
    }
}

int
dllistCheckCount(adllist)
    struct dllist    *adllist;
{
    struct dllist_node *tmpnode;
    int              fcount;
    int              bcount;

    if (adllist == NULL) {
        fprintf(stderr, "BadArgs to dllistCheckCount()\n");
        return (0);
    }
    fcount = 0;
    for (tmpnode = adllist->head; tmpnode != NULL; tmpnode = tmpnode->next)
        {
        fcount++;
    }
    bcount = 0;
    for (tmpnode = adllist->tail; tmpnode != NULL; tmpnode = tmpnode->prev)
        {
        bcount++;
    }
    return ((fcount == adllist->dllist_count) &&
        (bcount == fcount));
}

int
dllistCheckNode(adllist, node)
    struct dllist    *adllist;
    struct dllist_node *node;
{
    struct dllist_node *tmpnode;

    if ((adllist == NULL) || (node == NULL)) {
        fprintf(stderr, "BadArgs to dllistCheckNode()\n");
        return (0);
    }
    for (tmpnode = adllist->head; tmpnode != NULL; tmpnode = tmpnode->next)
        {
        if (tmpnode == node)
            return (1);
    }
    return (0);
}
```

```
struct dllist    *
dllistMakeNew()
{
    struct dllist    *new;

    new = (struct dllist *) malloc(sizeof(struct dllist));
    memset((char *) new, 0, sizeof(struct dllist));
    return (new);
}

struct dllist_node *
dllistMakeNewNode()
{
    struct dllist_node *new;

    new = (struct dllist_node *) malloc(sizeof(struct dllist_node));
    memset((char *) new, 0, sizeof(struct dllist_node));
    return (new);
}

void
dllistDestroy(adllist,fDestroyData)
    struct dllist    *adllist;
    int     fDestroyData;
{
    struct dllist_node *node, *nextNode;
    if (adllist == NULL) {
        fprintf(stderr, "BadArgs to dllistDestroy()\n");
        return;
    }
    /*
     * map (adllist, free);
     */
    for (node = adllist->head; node != NULL; ) {
        nextNode = node->next;
        if(fDestroyData) free(node->data);
        free(node);
        node = nextNode;
    }
    free(adllist);
    return;
}

void
dllistDestroyElements(adllist,fDestroyData)
    struct dllist    *adllist;
    int     fDestroyData;
{
    struct dllist_node *node, *nextNode;
    if (adllist == NULL) {
        fprintf(stderr, "BadArgs to dllistDestroyElements()\n");
        return;
    }
    for (node = adllist->head; node != NULL; ) {
        nextNode = node->next;
        if(fDestroyData) free(node->data);
        free(node);
        node = nextNode;
    }
    memset(adllist, 0, sizeof(struct dllist ));
    return;
}
void
dllistDestroyTail(adllist,aNode,fDestroyData)
    struct dllist      *adllist;
    struct dllist_node *aNode;
    int        fDestroyData;
{
    struct dllist_node *node, *nextNode;
    int i;
    if ((adllist == NULL) || (aNode == NULL )){
        fprintf(stderr, "BadArgs to dllistDestroyTail()\n");
        return;
    }
    for (node = aNode->next, i=0; node != NULL; i++) {
        nextNode = node->next;
        if(fDestroyData) free(node->data);
        free(node);
        node = nextNode;
    }
    adllist->dllist_count -= i;
    adllist->tail = aNode;
    return;
}
void
dllistInsertAtHead(adllist, node)
    struct dllist      *adllist;
    struct dllist_node *node;
{
    if ((adllist == NULL) || (node == NULL)) {
        fprintf(stderr, "BadArgs to dllistInsertAtHead()\n");
        return;
    }
    if (adllist->tail == NULL) {
        adllist->tail = node;
    }
    if(adllist->head != NULL){
    adllist->head->prev = node;
    }
    node->next = adllist->head;
    adllist->head = node;
    node->prev = NULL;
    adllist->dllist_count++;
    return;

}
```

```
void
dllistInsertAtTail(adllist, node)
    struct dllist    *adllist;
    struct dllist_node *node;
{
    if ((adllist == NULL) || (node == NULL)) {
        fprintf(stderr, "BadArgs to dllistInsertAtTail()\n");
        return;
    }
    if (adllist->head == NULL) {
        adllist->head = node;
    } else {
        adllist->tail->next = node;
    }
    node->next = NULL;
    node->prev = adllist->tail;
    adllist->tail = node;
    adllist->dllist_count++;
    return;

}

void
dllistInsertAfter(adllist, old, new)
    struct dllist    *adllist;
    struct dllist_node *old, *new;
{
    if ((adllist == NULL) || (old == NULL) || (new == NULL)) {
        fprintf(stderr, "BadArgs to dllistInsertAfter()\n");
        return;
    }
#ifdef CHECK_NODE
    if (!dllistCheckNode(adllist, node)) {
        fprintf(stderr, "node %ld not on dllist %ld\n", node, adllist);
        return;
    }
#endif              /* CHECK_NODE */
    adllist->dllist_count++;
    if (adllist->tail == old) {
        adllist->tail = new;
    }
    new->next = old->next;
    if(old->next){
        old->next->prev = new;
    }
    old->next = new;
    new->prev = old;
    return;
}


void
```

```
dllistInsertBefore(adllist, old, new)
    struct dllist    *adllist;
    struct dllist_node *old, *new;
{
    if ((adllist == NULL) || (old == NULL) || (new == NULL)) {
        fprintf(stderr, "BadArgs to dllistInsertBefore()\n");
        return;
    }
#ifdef CHECK_NODE
    if (!dllistCheckNode(adllist, node)) {
        fprintf(stderr, "node %ld not on dllist %ld\n", node, adllist);
        return;
    }
#endif              /* CHECK_NODE */
    adllist->dllist_count++;
    if (adllist->head == old) {
        adllist->head = new;
    }
    new->prev = old->prev;
    if(old->prev){
        old->prev->next = new;
    }
    old->prev = new;
    new->next = old;
    return;
}
void
dllistDeleteThis(adllist, node)
    struct dllist    *adllist;
    struct dllist_node *node;
{
    struct dllist_node *tmpnode;
    if ((adllist == NULL) || (node == NULL)) {
        fprintf(stderr, "BadArgs to dllistDeleteThis()\n");
        return;
    }
#ifdef CHECK_NODE
    if (!dllistCheckNode(adllist, node)) {
        fprintf(stderr, "node %ld not on dllist %ld\n", node, adllist);
        return;
    }
#endif              /* CHECK_NODE */
    adllist->dllist_count--;
    if (node == adllist->head) {
        adllist->head = node->next;
    }
    if (node == adllist->tail) {
        adllist->tail = node->prev;
    }
    if(node->prev != NULL){
        node->prev->next = node->next;
    }
    if(node->next != NULL){
        node->next->prev = node->prev;
    }
    /*free (node); *//* caller frees when he wants */
    return;
}

void
dllistMap(adllist, func, arg1, arg2)
    struct dllist    *adllist;
    void            (*func) ();
char           *arg1, *arg2;    /* space for args to func */
{
    struct dllist_node *node;
    if (adllist == NULL) {
        fprintf(stderr, "BadArgs to map()\n");
        return;
    }
    for (node = adllist->head; node != NULL; node = node->next) {
        func(node->data, arg1, arg2);
    }

}

void
dllistMapReverse(adllist, func, arg1, arg2)
    struct dllist    *adllist;
    void            (*func) ();
char           *arg1, *arg2;    /* space for args to func */
{
    struct dllist_node *node;
    if (adllist == NULL) {
        fprintf(stderr, "BadArgs to map()\n");
        return;
    }
    for (node = adllist->tail; node != NULL; node = node->prev) {
        func(node->data, arg1, arg2);
    }

}

void
dllistAppend(adllist, bdllist)  /* destructive append */
    struct dllist    *adllist, *bdllist;
{
    if ((adllist == NULL) || (bdllist == NULL)) {
        fprintf(stderr, "BadArgs to dllistAppend()\n");
        return;
    }
    if (adllist->tail == NULL) {
        memcpy(adllist, bdllist, sizeof(struct dllist));
    } else if (adllist->tail == NULL) {
    /*adllist is the result*/
    } else {
        adllist->tail->next = bdllist->head;
        bdllist->head->prev = adllist->tail;
        adllist->tail = bdllist->tail;
        adllist->dllist_count += bdllist->dllist_count;
    }
    memset(bdllist, 0, sizeof(struct dllist)); /* destruction */
    return;
}

void
dllistAfterInsertdlList(adllist, bdllist, node) /* destructive */
    struct dllist    *adllist, *bdllist;
    struct dllist_node *node;
{
    /* since node is on adllist, adllist->head won't be null */
    if ((adllist == NULL) || (bdllist == NULL) || (node == NULL)) {
        fprintf(stderr, "BadArgs to dllistAfterInsertdlList()\n");
        return;
    }
#ifdef CHECK_NODE
    if (!dllistCheckNode(adllist, node)) {
        fprintf(stderr, "node %ld not on dllist %ld\n", node, adllist);
        return;
    }
#endif              /* CHECK_NODE */
    if ((bdllist->head == NULL) || (bdllist->tail == NULL)) {
        memset(bdllist, 0, sizeof(struct dllist));
        return;     /* nothing changes */
    }
    adllist->dllist_count += bdllist->dllist_count;
    if (adllist->tail == node) {
        adllist->tail = bdllist->tail;
    }
    bdllist->tail->next = node->next;
    bdllist->head->prev = node;
    node->next = bdllist->head;
    return;
}

void
dllistBeforeInsertdlList(adllist, bdllist, node)    /* destructive */
    struct dllist    *adllist, *bdllist;
    struct dllist_node *node;
{
    /* since node is on adllist, adllist->head won't be null */
    if ((adllist == NULL) || (bdllist == NULL) || (node == NULL)) {
        fprintf(stderr, "BadArgs to dllistBeforeInsertdlList()\n");
        return;
    }
#ifdef CHECK_NODE
    if (!dllistCheckNode(adllist, node)) {
        fprintf(stderr, "node %ld not on dllist %ld\n", node, adllist);
        return;
    }
#endif              /* CHECK_NODE */
    if ((bdllist->head == NULL) || (bdllist->tail == NULL)) {
        memset(bdllist, 0, sizeof(struct dllist));
        return;     /* nothing changes */
    }
    adllist->dllist_count += bdllist->dllist_count;
    if (adllist->head == node) {
        adllist->head = bdllist->head;
    }
    bdllist->head->prev = node->prev;
    bdllist->tail->next = node;
    node->prev = bdllist->tail;
    return;
}

struct dllist_node
*dllistGetNthNode(adllist, n)
    struct dllist    *adllist;
    int n;
{
    int i;
    struct dllist_node  *node;

    if (adllist == NULL){
        fprintf(stderr, "BadArgs to dllistGetNthNode()\n");
        return NULL;
    }
    if ((n < 0) || (n > adllist->dllist_count)){
        return NULL;
    }
    else{
    for(i=0,node=adllist->head;i<n;i++,node=node->next){
    }
    return node;
    }
}

struct dllist_node
*dllistGetRevNthNode(adllist, n)
    struct dllist    *adllist;
    int n;
{
    int i;
    struct dllist_node  *node;

    if (adllist == NULL){
        fprintf(stderr, "BadArgs to dllistGetRevNthNode()\n");
        return NULL;
    }
    if ((n < 0) || (n > adllist->dllist_count)){
        return NULL;
    }
    else{
    for(i=0,node=adllist->tail;i<n;i++,node=node->prev){
    }
    return node;
    }
}

int
dllistSize(adllist)
    struct dllist    *adllist;
{
    struct dllist_node *node;
    int i;
    if (adllist == NULL) {
        fprintf(stderr, "BadArgs to map()\n");
        return -1;
    }
    for (node = adllist->head,i=0; node != NULL; node = node->next,i++) {
    }
    return i;

}
```

```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/*
 *   hash.c hash table routines
 *
 *   author -- Vinod Anupam
 *
 *   modification history
 *
 *  Hash Table & Symbol management routines
 */

#include <stdio.h>
#include <string.h>

#include <shastra/shilp.h>
#include <shastra/utils/hash.h>

#define HASH_TALK

/*
 * htHashFunxnBytes(sb,n,prime)  --- compute hash value of n bytes at sb
 */
int     htHashFuncBytes (sb,n,prime)
char   *sb;
int n;
int prime;
{
    int i;
    unsigned    ch = 0,
                chTemp;
    for (i=0; i<n;i++){
    ch = (ch << 4) + (*sb++);
    if (chTemp = ch & 0xf0000000) {
        ch = ch ^ (chTemp >> 24);
        ch = ch ^ chTemp;
    }
    }
    return (ch % prime);
}
/*
 * htHashFunxnSb(sb,prime)  --- compute hash value of sb
 */
int     htHashFuncSb (sb,prime)
char   *sb;
int prime;
{

    char   *sbTemp;
    unsigned    ch = 0,
                chTemp;
    for (sbTemp = sb; *sbTemp != fEndOfString; sbTemp++) {
    ch = (ch << 4) + (*sbTemp);
    if (chTemp = ch & 0xf0000000) {
        ch = ch ^ (chTemp >> 24);
        ch = ch ^ chTemp;
    }
    }
    return (ch % prime);
}


/*
 * htLookup(ht,sb)   ---- lookup sb in the hash table
 */
struct he  *htLookup (pht,sb)
hashTable *pht;
char   *sb;
{
    int     ihe;
    struct he  *phe;

    if(pht->iElementSize){
        ihe = pht->hashFunc(sb,pht->iElementSize,pht->ihtSize);
        for (phe = pht->rgphe[ihe]; phe != NULL; phe = phe -> phe) {
            if (memcmp (sb, phe -> sb, pht->iElementSize) == 0){
                return (phe);
            }
        }
    }
    else{
        ihe = pht->hashFunc(sb,pht->ihtSize);
        for (phe = pht->rgphe[ihe]; phe != NULL; phe = phe -> phe) {
            if (strcmp (sb, phe -> sb) == 0){
                return (phe);
            }
        }
    }
    return (NULL);
}

/*
 * htInstallSymbol(pht,sb,data)   ----- install sb in the hash table
 */
struct he  *htInstallSymbol (pht,sb,data)
hashTable *pht;
char  *sb;
char  *data;
{
    struct he  *phe,*pheS;
    int     ihe;

    phe = htLookup (pht,sb);
    if (phe == NULL) {      /* not in table */
        phe = heGet ();
        if(pht->iElementSize){
            phe -> sb = htMakeBytes(sb,pht->iElementSize);
            ihe = pht->hashFunc (sb, pht->iElementSize, pht->ihtSize);
        }
        else{
            phe -> sb = htMakeString(sb);
            ihe = pht->hashFunc (sb, pht->ihtSize);
        }
        phe -> phe = pht->rgphe[ihe];
        pht->rgphe[ihe] = phe;
        phe -> pheGroup = pht->pheStart;
        phe->data = data;
        pht->pheStart = phe;

    }
    /*symbol installed in table only once*/
    return phe;
}


/*
 * htMakeBytes(sb,n)   ---create a copy of n bytes sb
 */
char   *htMakeBytes (sb,n)
char   *sb;
int n;
{
    char   *sbNew;
    sbNew = (char*)malloc(n);
    memcpy (sbNew,sb, n);
    return (sbNew);
}
```

```
/*
 * htMakeString(sb)   ---create a copy of string sb
 */
char   *htMakeString (sb)
char   *sb;
{
    char   *sbNew;
    sbNew = strdup(sb);
    return (sbNew);
}



/*
 * htMakeNew(iSize,iEltSize)  -----prepares the hash table initially
 * iSize must be a prime no < iheMax
 * iEltSize must be 0 for variable size, else element size
 */
hashTable *htMakeNew (iSize,iEltSize)
int iSize;
int iEltSize;
{

    int     ihe;
    hashTable * pht;

    pht = (hashTable *)malloc(sizeof(hashTable));
    for (ihe = 0; ihe < iheMax; ihe++){
        pht->rgphe[ihe] = NULL;
    }
    pht->pheStart = NULL;
    pht->ihtSize = iSize;
    pht->iElementSize = iEltSize;
    if(iEltSize){
        pht->hashFunc = htHashFuncBytes;
    }
    else{
        pht->hashFunc = htHashFuncSb;
    }

    return(pht);
}


/*
 * heDelete(pht,sb)   ---- delete this entry sb from the hash table
 */
struct he  *heDelete (pht,sb)
hashTable * pht;
char   *sb;
{
    int     ihe;
    struct he  *phe,
               *pheFollow;

    if(pht->iElementSize){
        ihe = pht->hashFunc (sb, pht->iElementSize, pht->ihtSize);
        pheFollow = pht->rgphe[ihe];
        for (phe = pheFollow; phe != NULL; phe = phe -> phe) {
            if (memcmp (sb, phe -> sb, pht->iElementSize) == 0) {
                break;
            }
            else {
                pheFollow = phe;
            }
        }
    }
    else{
        ihe = pht->hashFunc (sb, pht->ihtSize);
        pheFollow = pht->rgphe[ihe];
        for (phe = pheFollow; phe != NULL; phe = phe -> phe) {
            if (strcmp (sb, phe -> sb) == 0) {
                break;
            }
            else {
                pheFollow = phe;
            }
    }
    }
    if (phe == NULL) {
    printf("heDelete : Can't find it in hash table!\n");
    return (NULL);
    }
    if (pheFollow != phe) {
    pheFollow -> phe = phe -> phe;/* delete from ll */
    }
    else{
    pht->rgphe[ihe] = NULL;
    }
    if(pht->pheStart == phe){
        pht->pheStart = phe->pheGroup;
    }
    else{
        for (pheFollow=pht->pheStart; pheFollow->pheGroup != phe;
                pheFollow = pheFollow -> pheGroup) {
        }
        pheFollow->pheGroup = phe->pheGroup;
    }
    return (phe); /*this is being removed*/
}


/*
 * heGet() ---- returns a he from memory
 */
```

```
struct he  *heGet () {
    struct he  *phe;

    phe = (struct he *)malloc(sizeof(struct he));
    phe -> sb = NULL;
    phe -> phe = NULL;
    phe -> pheGroup = NULL;
    return phe;
}


/*
 * htDestroy()  ---- destroy a hash table and contents.. if fRec, destroy
     data
 */
void htDestroy (pht,fRecurse)
hashTable *pht;
int     fRecurse;           /* 1 destroy data */
{
    struct he  *phe, *ophe;;

    for (phe = pht->pheStart; phe != NULL; ){
        ophe = phe;
        phe = phe -> pheGroup;
        if(heDelete(pht,ophe->sb) == NULL){
            fprintf(stderr,"htDestroy()-> internal error on %s!\n",
                ophe->sb);
        }
        if(fRecurse){
            free(ophe -> data);
        }
        free(ophe);
    }
    free(pht);
}



/*
 * htDump()  ---- dumps contents of hash table in order of entry
 */
void htDump (pht,mode)
hashTable *pht;
int     mode;           /* 0 insertion 1 hashed */
{
    struct he  *phe;
    int     ihe;

    printf ("Dumping hash in mode %d\n", mode);
    if (mode) {
    for (ihe = 0; ihe < pht->ihtSize; ihe++) {
        for (phe = pht->rgphe[ihe]; phe != NULL; phe = phe -> phe) {
        printf ("%ld : %s\n", phe -> sb, phe -> data);
        }
    }
    }
    else {
    for (phe = pht->pheStart; phe != NULL; phe = phe -> pheGroup)
        printf ("%ld : %s\n", phe -> sb, phe -> data);
    }

}

#define NOHASH_STANDALONE
#ifdef HASH_STANDALONE
/*
 * test.c
 */

char    *hash_str[] = {
    "1",        "one",
    "2",        "two",
    "3",        "three",
    "4",        "four",
    "one",      "1",
    "two",      "2",
    "three",    "3",
    "four",     "4"
};
#define MAXENTCOUNT 16


struct testdata{
long ent;
char* val;
} test[] ={
    1,      "one",
    111,        "two",
    2323,       "three",
    24,     "four",
    1212,       "five",
    65536,      "six"
};

#define MAXTSTCOUNT 6
main()
{
hashtest2();
}

hashtest1(){
    hashTable* pht;
    int ihe;
    struct he *phe;

    printf("Hello Hasho !\n");
    pht = htMakeNew(31,0); /*31 entries,variable size*/

 /*    install temp data */
    for (ihe = 0; ihe < MAXENTCOUNT; ihe += 2) {
    htInstallSymbol (pht,hash_str[ihe], hash_str[ihe + 1]);
    }
    htDump(pht,0);
    htDump(pht,1);
    for (ihe = 0; ihe < MAXENTCOUNT; ihe += 2) {
        phe = htLookup (pht,hash_str[ihe]);
        printf ("%s (looked up)-> %s\n", phe -> sb, phe -> data);
    }
    phe = heDelete(pht,"three");
    printf ("%s (deleted)-> %s\n", phe -> sb, phe -> data);
    htDump(pht,0);
    htDump(pht,1);
    for (ihe = 0; ihe < MAXENTCOUNT; ihe += 2) {
        phe = htLookup (pht,hash_str[ihe]);
        if(phe!=NULL){
        printf ("%s (looked up)-> %s\n", phe -> sb, phe -> data);
        }
   }
}
hashtest2(){
    hashTable* pht;
    int ihe;
    struct he *phe;

    printf("Hello Hasho !\n");
    pht = htMakeNew(31,sizeof(long)); /*31 entries,sizeof(long)size*/

 /*    install temp data */
    for (ihe = 0; ihe < MAXTSTCOUNT; ihe ++) {
    htInstallSymbol (pht,(char *)&test[ihe].ent, test[ihe ].val);
    }
    htDump(pht,0);
    htDump(pht,1);
    for (ihe = 0; ihe < MAXTSTCOUNT; ihe ++ ) {
        phe = htLookup (pht,(char *)&test[ihe].ent);
        printf ("%ld (looked up)-> %s\n", phe -> sb, phe -> data);
    }
    phe = heDelete(pht,(char*)&test[2].ent);
    printf ("%ld (deleted)-> %s\n", phe -> sb, phe -> data);
    htDump(pht,0);
    htDump(pht,1);
    for (ihe = 0; ihe < MAXTSTCOUNT; ihe ++) {
        phe = htLookup (pht,(char *)&test[ihe].ent);
        if(phe!=NULL){
        printf ("%ld (looked up)-> %s\n", phe -> sb, phe -> data);
        }
    }
}
```

```
#endif /*HASH_STANDALONE*/
```

```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
#include <stdio.h>
#include <malloc.h>

#include <shastra/utils/list.h>

int
listCheckGood(alist)
    struct list    *alist;
{
    int             badlist = 1;

    if (alist == NULL) {
        fprintf(stderr, "BadArgs to listCheckGood)\n");
        return (0);
    }
    if (alist->head == NULL) {
        if (alist->tail == NULL) {
            if (alist->list_count != 0) {
                badlist = 0;
            }
        } else {
            badlist = 0;
        }
    } else {
        if (alist->tail == NULL) {
            badlist = 0;
        }
    }
    if (!badlist) {
        return 0;
    } else {
        return listCheckCount(alist);
    }
}

int
listCheckCount(alist)
    struct list    *alist;
{
    struct list_node *tmpnode;
    int             count;

    if (alist == NULL) {
        fprintf(stderr, "BadArgs to listCheckCount()\n");
        return (0);
    }
    count = 0;
    for (tmpnode = alist->head; tmpnode != NULL; tmpnode = tmpnode->next) {
        count++;
    }
    return (count == alist->list_count);
}

int
listCheckNode(alist, node)
    struct list    *alist;
    struct list_node *node;
{
    struct list_node *tmpnode;

    if ((alist == NULL) || (node == NULL)) {
        fprintf(stderr, "BadArgs to listCheckNode()\n");
        return (0);
    }
    for (tmpnode = alist->head; tmpnode != NULL; tmpnode = tmpnode->next) {
        if (tmpnode == node){
            return (1);
        }
    }
    return (0);
}

int
listGetNodeIndex(alist, data)
    struct list    *alist;
    char *data;
{
    struct list_node *tmpnode;
    int i;

    if (alist == NULL){
        fprintf(stderr, "BadArgs to listGetNodeIndex()\n");
        return (-1);
    }
    for (i=0,tmpnode = alist->head; tmpnode != NULL;
        tmpnode = tmpnode->next, i++) {
        if (tmpnode->data == data){
            return (i);
        }
    }
    return (-1);
}

struct list_node *
listFindNode(alist, data)
    struct list    *alist;
    char *data;
{
    struct list_node *tmpnode;

    if (alist == NULL){
        fprintf(stderr, "BadArgs to listFindNode()\n");
        return (NULL);
    }
    for (tmpnode = alist->head; tmpnode != NULL; tmpnode = tmpnode->next) {
        if (tmpnode->data == data){
            return (tmpnode);
        }
    }
    return (NULL);
}

struct list    *
listMakeNew()
{
    struct list    *new;

    new = (struct list *) malloc(sizeof(struct list));
    memset((char *) new, 0, sizeof(struct list));
    return (new);
}

struct list_node *
listMakeNewNode()
{
    struct list_node *new;

    new = (struct list_node *) malloc(sizeof(struct list_node));
    memset((char *) new, 0, sizeof(struct list_node));
    return (new);
}

void
listDestroy(alist,fDestroyData)
    struct list    *alist;
    int     fDestroyData;
{
    struct list_node *node, *next_node;
    if (alist == NULL) {
        fprintf(stderr, "BadArgs to listDestroy()\n");
        return;
    }
    /*
     * map (alist, free);
     */
    for (node = alist->head; node != NULL; ) {
        next_node = node->next;
        if(fDestroyData && (node->data != NULL)){
            free(node->data);
        }
        free(node);
        node = next_node;
    }
    free(alist);
    return;
}

void
listDestroyElements(alist,fDestroyData)
    struct list    *alist;
    int     fDestroyData;
{
    struct list_node *node, *next_node;
    if (alist == NULL) {
        fprintf(stderr, "BadArgs to listDestroyElements()\n");
        return;
    }
    for (node = alist->head; node != NULL; ) {
        next_node = node->next;
        if(fDestroyData && (node->data != NULL)){
            free(node->data);
        }
        free(node);
        node = next_node;
    }
    memset(alist, 0, sizeof(struct list ));
    return;
}
void
listInsertAtHead(alist, node)
    struct list    *alist;
    struct list_node *node;
{
    if ((alist == NULL) || (node == NULL)) {
        fprintf(stderr, "BadArgs to listInsertAtHead()\n");
        return;
    }
    if (alist->tail == NULL) {
        alist->tail = node;
    }
    node->next = alist->head;
    alist->head = node;
    alist->list_count++;
    return;

}

void
listInsertAtTail(alist, node)
    struct list     *alist;
    struct list_node *node;
{
    if ((alist == NULL) || (node == NULL)) {
        fprintf(stderr, "BadArgs to listInsertAtTail()\n");
        return;
    }
    if (alist->head == NULL) {
        alist->head = node;
    } else {
        alist->tail->next = node;
    }
    alist->tail = node;
    node->next = NULL;
    alist->list_count++;
    return;

}

void
listInsertAfter(alist, old, new)
    struct list     *alist;
    struct list_node *old, *new;
{
    if ((alist == NULL) || (old == NULL) || (new == NULL)) {
        fprintf(stderr, "BadArgs to listInsertAfter()\n");
        return;
    }
#ifdef CHECK_NODE
    if (!listCheckNode(alist, node)) {
        fprintf(stderr, "node %ld not on list %ld\n", node, alist);
        return;
    }
#endif              /* CHECK_NODE */
    alist->list_count++;
    if (alist->tail == old) {
        alist->tail = new;
    }
    new->next = old->next;
    old->next = new;
    return;
}
```

```
void
listDeleteThis(alist, node)
    struct list    *alist;
    struct list_node *node;
{
    struct list_node *tmpnode;
    if ((alist == NULL) || (node == NULL)) {
        fprintf(stderr, "BadArgs to listDeleteThis()\n");
        return;
    }
#ifdef CHECK_NODE
    if (!listCheckNode(alist, node)) {
        fprintf(stderr, "node %ld not on list %ld\n", node, alist);
        return;
    }
#endif              /* CHECK_NODE */
    alist->list_count--;
    if (node == alist->head) {
        alist->head = node->next;
        if (node == alist->tail) {
            alist->tail = NULL;
        }
    }
    else{
        for(tmpnode = alist->head;tmpnode->next != node;tmpnode=tmpnode->
            next){
        } /*get to prev node*/
        tmpnode->next = node->next;
        if (node == alist->tail) {
        alist->tail = tmpnode;
        }
    }
    /*free (node); *//* caller frees when he wants */
    return;
}

void
listDeleteThisData(alist, data)
    struct list    *alist;
    char *data;
{
    struct list_node *tmpnode;
    if (alist == NULL){
        fprintf(stderr, "BadArgs to listDeleteThisData()\n");
        return;
    }
    tmpnode = listFindNode(alist,data);
    if(tmpnode != NULL){
        listDeleteThis(alist, tmpnode);
        free (tmpnode);
    }
    return;
}

void
listMap(alist, func, arg1, arg2)
    struct list    *alist;
    void            (*func) ();
char           *arg1, *arg2;    /* space for args to func */
{
    struct list_node *node;
    if (alist == NULL) {
        fprintf(stderr, "BadArgs to map()\n");
        return;
    }
    for (node = alist->head; node != NULL; node = node->next) {
        func(node->data, arg1, arg2);
    }

}

void
listAppend(alist, blist)    /* destructive append */
    struct list    *alist, *blist;
{
    if ((alist == NULL) || (blist == NULL)) {
        fprintf(stderr, "BadArgs to listAppend()\n");
        return;
    }
    if (alist->tail == NULL) {
        memcpy(alist, blist, sizeof(struct list));
    } else if (blist->tail == NULL) {
    /*alist unchanged*/
    } else {
        alist->tail->next = blist->head;
        alist->tail = blist->tail;
        alist->list_count += blist->list_count;
    }
    memset(blist,  0, sizeof(struct list));     /* destruction */
    return;
}

void
listAfterInsertList(alist, blist, node) /* destructive */
    struct list    *alist, *blist;
    struct list_node *node;
{
    /* since node is on alist, alist->head won't be null */
    if ((alist == NULL) || (blist == NULL) || (node == NULL)) {
        fprintf(stderr, "BadArgs to listAfterInsertList()\n");
        return;
    }
#ifdef CHECK_NODE
    if (!listCheckNode(alist, node)) {
        fprintf(stderr, "node %ld not on list %ld\n", node, alist);
        return;
    }
#endif              /* CHECK_NODE */
    if ((blist->head == NULL) || (blist->tail == NULL)) {
        memset(blist, 0, sizeof(struct list));
        return;     /* nothing changes */
    }
    alist->list_count += blist->list_count;
    if (alist->tail == node) {
        alist->tail = blist->tail;
    }
    blist->tail->next = node->next;
    node->next = blist->head;
    return;
}

struct list_node
*listGetNthNode(alist, n)
    struct list    *alist;
    int n;
{
    int i;
    struct list_node  *node;

    if (alist == NULL){
        fprintf(stderr, "BadArgs to listGetNthNode()\n");
        return NULL;
    }
    if ((n < 0) || (n > alist->list_count)){
        return NULL;
    }
    else{
    for(i=0,node=alist->head;i<n;i++,node=node->next){
    }
    return node;
    }
}

int
listSize(alist)
    struct list    *alist;
{
    struct list_node *node;
    int i;
    if (alist == NULL) {
        fprintf(stderr, "BadArgs to map()\n");
        return -1;
    }
    for (node = alist->head,i=0; node != NULL; node = node->next,i++) {
    }
    return i;
    }
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
#include <stdio.h>

/* a more robust interface to malloc and free */
#include <malloc.h>

char           *
memMalloc(c)
    int             c;
{
    char           *temp;
    if(c <=0){
         fprintf(stderr, "memMalloc()->Warning: trying to malloc %d!\n",c);
         return NULL;
    }
    if(c < 32){
        c = 32;
    }
    temp = malloc((unsigned) c);
    if (temp == NULL) {
        fprintf(stderr, "memMalloc()->Out of memory. Wanted %d\n",c);
        exit(-1);
    } else{
        return temp;
    }
}

char           *
memCalloc(size, num)
    int             size;
    int                 num;
{
    char               *temp;
    if((size <=0)||(num <=0)){
         fprintf(stderr, "memCalloc()->Warning: trying to calloc %d,%d!\n",
        size,num);
         return NULL;
    }
    temp = calloc((unsigned) size, num);
    if (temp == NULL) {
        fprintf(stderr, "memCalloc()->Out of memory.Wanted %d,%d\n",
                        size,num);
        exit(-1);
    } else
        return temp;
}

char               *
memRealloc(p, num)
    char               *p;
    int                 num;
{
    char               *temp;
    if(num <=0){
         fprintf(stderr, "memRealloc()->Warning: trying to realloc %d!\n",
        num);
         return NULL;
    }
    if(num < 32){
        num = 32;
    }
    temp = realloc(p, (unsigned) num);
    if (temp == NULL) {
        fprintf(stderr, "memRealloc()->Out of memory.Wanted %d\n", num);
        exit(-1);
    } else
        return temp;
}

void
memFreeMem(p)
char *p;
{
    if(p != NULL){
        free(p);
    }
    else{
        fprintf(stderr, "Warning.. freeing NULL!\n");
    }
}

void
memTest()
```

```
{
int i;
char* p;
    printf("memTest()->doing some checks!\n");
    for(i=1;i<1024;i++){
        p = memMalloc(i);
        memFreeMem(p);
    }
    printf("memTest()->done !\n");
}
```

```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
#include <stdio.h>

#define INIT   register char *sp = instring;
#define GETC() (*sp++)
#define PEEKC()     (*sp)
#define UNGETC(c)   (--sp)
#define RETURN(c)   return;
#define ERROR(c)    regError(c)

#include <regexp.h>
#include <shastra/utils/regExpr.h>

#define DEBUG
#define STANDALONEnn

void
compileRegExp(regExpr, regBufStart, regBufSize)
    char           *regExpr;
    char           *regBufStart;
    int             regBufSize;
{
    /*
     * char *compile(instring, expbuf, endbuf, eof)
     */
    (void) compile(regExpr, regBufStart, &regBufStart[regBufSize], '\0');
#ifdef DEBUG
    printf("compileRegExp()-> compiled %s to %s\n",
           regExpr, regBufStart);
#endif
```

```
}

int
matchRegExp(dataString, regExpBuf)
    char           *dataString;
    char           *regExpBuf;
{
    /*
     * int step(string, expbuf)
     */
    return (step(dataString, regExpBuf));
}


regError(c)
    int             c;
{
    fprintf(stderr, "regError(): ");
    switch (c) {
    case 11:
        fprintf(stderr,"Range endpoint too large.\n");
        break;
    case 16:
        fprintf(stderr,"Bad number.\n");
        break;
    case 25:
        fprintf(stderr,"``\ digit'' out of range.\n");
        break;
    case 36:
        fprintf(stderr,"Illegal or missing delimiter.\n");
        break;
    case 41:
        fprintf(stderr,"No remembered search string.\n");
        break;
    case 42:
        fprintf(stderr,"\( \) imbalance.\n");
        break;
    case 43:
        fprintf(stderr,"Too many \(.\n");
        break;
    case 44:
        fprintf(stderr,"More than 2 numbers  given  in \{ \}.\n");
        break;
    case 45:
        fprintf(stderr,"} expected after \.\n");
        break;
    case 46:
        fprintf(stderr,"First number exceeds second in \{ \}.\n");
        break;
    case 49:
        fprintf(stderr,"[] imbalance.\n");
        break;
    case 50:
        fprintf(stderr,"Regular expression too long.\n");
        break;
    }
}



#ifdef STANDALONE
main()
{
#define ESIZE 256
    char            expbuf[ESIZE];
    char            inbuf[256];
    int             i;
static char *mptnsb[] = { "",
    "ABSOLUTE",
    "BOOHOO",
    "CHARACTER",
    "DISTINCT",
    "EUPHORIA",
    "FIRST",
    "GO",
    "HEGEMONY",
    "INDICATOR",
    "JOCULAR",
    "KNAPSACK",
    "LANGUAGE",
    "MODULE",
    "NAME",
    "ON",
    "PRECISION",
    "QUARTZ",
    "RESTRICT",
    "SECTION",
    "TUMBLEWEED",
    "UNIQUE",
    "VALUES",
    "WHENEVER",
    "XCITING",
    "YEOMAN",
    "ZEBRA" };

    while (gets(inbuf) != NULL) {
        compileRegExp(inbuf, expbuf, ESIZE);

        for (i = 0; i < 26; i++) {
            if (matchRegExp(mptnsb[i], expbuf))
                printf("%s matched \t", mptnsb[i], inbuf);
        }
    }
}
#endif              /* STANDALONE */
```

```
/**************************************************************************
    ***/
/**************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/**************************************************************************
    ***/
/**************************************************************************
    ***/
#include <stdio.h>

#include <shastra/shilp.h>
#include <shastra/utils/tree.h>

/* binary trees */

struct tree    *
treeMakeNew(data)
    int             data;
{
    struct tree    *new;

    new = (struct tree *) malloc(sizeof(struct tree));
    new->left = NULL;
    new->right = NULL;
    new->parent = NULL;
    new->control = 0;
    new->data = 0;
    return (new);
}

void
treeInorder(atree, func)
    struct tree    *atree;
    void            (*func) ();
{
    if (atree == NULL) {
        return;
    }
    if (atree->left != NULL) {
        treeInorder(atree->left, func);
    }
    func(atree);        /* func applied at node */
    if (atree->right != NULL) {
        treeInorder(atree->right, func);
    }
    return;
}

void
treePreorder(atree, func)
    struct tree    *atree;
    void            (*func) ();
{
    if (atree == NULL) {
        return;
    }
    func(atree);        /* func applied at node */
    if (atree->left != NULL) {
        treePreorder(atree->left, func);
    }
    if (atree->right != NULL) {
        treePreorder(atree->right, func);
    }
    return;
}

void
treePostorder(atree, func)
    struct tree    *atree;
    void            (*func) ();
{
    if (atree == NULL) {
        return;
    }
    if (atree->left != NULL) {
        treePostorder(atree->left, func);
    }
    if (atree->right != NULL) {
        treePostorder(atree->right, func);
    }
    func(atree);        /* func applied at node */
    return;
}

struct tree    *
treeInsert(atree, data)
    struct tree    *atree;
    int             data;
{
    struct tree    *node;
    if (atree == NULL) {
        fprintf(stderr, "BadArg to insert(%ld,%d)\n", atree, data);
        return NULL;
    }
    if (data == atree->data) {
        return (atree); /* nilpo duplication */
    } else if (data < atree->data) {
        if (atree->left == NULL) {
            atree->left = node = treeMakeNew(data);
            node->parent = atree;
            return (node);
        } else {
            return (treeInsert(atree->left, data));
        }
    } else {
        if (atree->right == NULL) {
            atree->right = node = treeMakeNew(data);
            node->parent = atree;
            return (node);
        } else {
            return (treeInsert(atree->right, data));
        }
    }
}

struct tree    *
treeBinarySearch(atree, data)
    struct tree    *atree;
    int             data;
{
    if (atree == NULL) {
        return NULL;
    }
    if (data == atree->data) {
        return (atree); /* found */
    } else if (data < atree->data) {
        return (treeBinarySearch(atree->left, data));
    } else {
        return (treeBinarySearch(atree->right, data));
    }
}

struct tree    *
treeFindNextSmaller(atree)
struct tree *atree;
{
    struct tree    *node;

    if ((node = atree->left) == NULL) {
        return (NULL);
    }
    for (node; node->right != NULL; node = node->right) {
    }
    return (node);
}

struct tree    *
treeFindNextBigger(atree)
struct tree *atree;
{
    struct tree    *node;

    if ((node = atree->right) == NULL) {
        return (NULL);
    }
    for (node; node->left != NULL; node = node->left) {
    }
    return (node);
}

void
treeDeleteThis(atree, node)
    struct tree    *atree, *node;
{
    struct tree    *nbor;

    if ((nbor = treeFindNextBigger(atree)) == NULL) {

    } else {
        nbor->parent->left = NULL;
        nbor->left = atree->left;
        nbor->parent = atree->parent;
        if (atree->parent == NULL) {    /* deleting root */
        } else {
                if (check_am_lsub(atree)) {
                atree->parent->left = nbor;
                }
            else {
                atree->parent->right = nbor;
            }
        }
    }
}
```